UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

ESCOLA DE ENGENHARIA

Programa de Pós-Graduação em Engenharia de Minas, Metalúrgica e de Materiais - PPGEM

ARQUITETURA DE HARDWARE DE BAIXO CUSTO PARA SISTEMAS TEMPO-REAL DISTRIBUÍDOS

MOISÉS DE MOURA BEHAR PONTREMOLI

Dissertação para obtenção do título de Mestre em Engenharia

Porto Alegre

1998

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

ESCOLA DE ENGENHARIA

Programa de Pós-Graduação em Engenharia de Minas, Metalúrgica e de Materiais - PPGEM

ARQUITETURA DE HARDWARE DE BAIXO CUSTO PARA SISTEMAS TEMPO-REAL DISTRIBUÍDOS

MOISÉS DE MOURA BEHAR PONTREMOLI

Engenheiro Eletricista

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Minas, Metalúrgica e de Materiais - PPGEM, como parte dos requisitos para a obtenção do título de Mestre em Engenharia.
Área de concentração: Instrumentação Eletro-Eletrônica.
Desenvolvida no Laboratório de Instrumentação Eletro-Eletrônica do Departamento de Engenharia Elétrica da UFRGS.

Porto Alegre

1998

ARQUITETURA DE HARDWARE DE BAIXO CUSTO PARA SISTEMAS TEMPO-REAL DISTRIBUÍDOS

MOISÉS DE MOURA BEHAR PONTREMOLI

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador:_____

Prof. Carlos Eduardo Pereira - UFRGS

Doutor em Engenharia pela Universidade de Stuttgart

Banca Examinadora:

Prof. Altamiro Amadeu Suzim, DELET/UFRGS

Doutor em Informática pelo INPG - França

Prof. João César Netto, II/UFRGS

Doutor em Ciências Aplicadas pela UCL - Bélgica

Prof. Renato Machado de Brito, DELET/UFRGS

Doutor em Engenharia pela UFRGS

Coordenador do PPGEM:_____

Prof. Dr. Luis Frederico Pinheiro Dick

Porto Alegre, Dezembro de 1998.

Dedico este trabalho aos meus pais Moisés e Eleusa, a minha namorada Claudia, a minha irmã Mila e ao meu amigo Sandro Moretti, pois foram e são as pessoas que mais influenciam para que eu siga em frente nesta eterna busca de aprendizado, conquistas e felicidade!

AGRADECIMENTOS

A todos que colaboraram direta ou indiretamente na elaboração deste trabalho, o meu reconhecimento.

Ao Professor Carlos Eduardo Pereira pela orientação incansável, tutoriamento e cobranças necessárias, empenho em fornecer os conhecimentos e infra-estrutura necessária para que o trabalho fosse feito e principalmente pela demonstração de garra e empenho na busca de um ideal, demonstrando com cada ato que é possível fazer um Brasil melhor para todos.

Ao Professor Renato Machado de Brito pelos conselhos e orientação que sempre me proporcionou desde o tempo da graduação em Engenharia Elétrica. Seus conselhos foram fundamentais para eu ingressar e realizar com tanto ganho o curso de pós-graduação que culmina com este trabalho.

Ao Engenheiro Sandro Rafael dos Santos que tanta dedicação despendeu para que este trabalho tivesse o máximo de êxito possível, tanto enquanto bolsista de Iniciação Científica quanto após a sua graduação, sacrificando noites e fins-de-semana para este fim.

Aos bolsistas de Iniciação Científica Marcelo Gotz e Charles André Carvalho que contribuíram na implementação do hardware que trata este trabalho.

Aos colegas Ronaldo Husemann e Mírian Cáceres que forneceram informações necessárias para que este trabalho tivesse seqüência.

Ao Engenheiro Edson Schuler e à Máquinas Kehl S.A. que forneceram material sobre o núcleo de Tempo-Real por eles desenvolvidos para que fosse possível realizar as comparações que são apresentadas no decorrer deste trabalho.

A todos os colegas do Laboratório de Instrumentação Eletro-Eletrônica da UFRGS pelas sugestões, observações valiosas e amizades que sempre me propiciaram.

Aos professores que tive o prazer de receber ensinamentos durante o curso de mestrado que culmina com este trabalho, quer seja em aulas formais ou em consultas informais que sempre foram atendidas com presteza e boa vontade.

Aos funcionários do PPGEM e IEE que deram suporte sempre que necessário.

Ao CNPq que forneceu o suporte financeiro para que todo este trabalho fosse possível de ser realizado.

SUMÁRIO

LISTA DE FIGURAS

RESUMO

Sistemas em tempo real caracterizam-se como tal quando seu correto funcionamento depende não apenas do correto processamento lógico de entradas e saídas, mas também da observância de restrições temporais na geração dos sinais de saída.

Assim sendo, uma das características principais de dispositivos usados em aplicações em tempo real é seu determinismo, ou seja, sua capacidade de responder, em qualquer circunstância, dentro de limites de tempo previamente determinados. Sendo esta característica um requisito básico, percebe-se a vantagem de dispor de um *hardware* que a incorpore. Facilitando o desenvolvimento de aplicações para esse tipo de controle, com um melhor desempenho.

Visando atender esta necessidade, este trabalho apresenta uma arquitetura de *hardware* de baixo custo para desenvolvimento de aplicações com requisitos de tempo-real para sistemas de controle distribuído.

Cada unidade de processamento da arquitetura distribuída é formada pelos seguintes elementos:

- Processador principal:
- Gerenciador de *timer* e de tarefas: responsável pela gerência de *timer* e interrupções, bem como pela gerência dos instantes de ativação das tarefas concorrentes e sincronização com outras unidades de processamento do sistema distribuído de automação.
- Processador de comunicação: responsável pela comunicação entre tarefas, incluindo o mapeamento para o protocolo de comunicação usado no barramento industrial (como por exemplo o Profibus).
- Periféricos, tais como memórias, co-processadores aritméticos, unidades de disco, etc.

A principal novidade proposta é o aproveitamento do baixo custo dos microcontroladores comerciais, atribuindo funções específicas para cada um, deixando o peso

computacional do sistema operacional, na sua maior parte, em processadores diferentes do responsável pelo processamento das tarefas da aplicação.

Dentre as vantagens da arquitetura proposta cabe aqui mencionar:

- Aumento do poder de processamento de uma Unidade de Processamento das tarefas da aplicação do usuário.

- Maior facilidade em obter o determinismo temporal, característica fundamental em sistemas tempo-real distribuídos.

- Possibilidade de utilização de algoritmos de escalonamento mais complexos e especializados, sem uma sobrecarga proibitiva no desempenho do sistema.

Os primeiros resultados obtidos com esta arquitetura, quando comparados com o exemplo industrial utilizado[1], são promissores. A recepção da comunidade científica também foi positiva, fato que pode ser medido pela aceitação dos diversos artigos apresentados e/ou publicados que basearam-se na arquitetura proposta nesta dissertação, quais sejam o 4th IFAC Workshop on Algorithms and Architectures for Real-Time Control [PoPe97b], Special Section of IFAC Control Engineering Practice Journal [PoPe97c], Tercer Taller Iberoamericano de Microeletrónica y sus Aplicaciones [Souza97], Euromicro'97 Workshop on Real-Time Systems [Parisoto97], Seminário Interno do DELET e IEE [PoPe96], Revista Egatea [PoPe97a] e o XII Congresso Brasileiro de Automática [PPS98].

[1] Descrito no capítulo 8.

ABSTRACT

This work presents a low-cost *hardware* architecture that enhances the performance and increases the predictability of real-time distributed systems. The proposed architecture overcomes one of the major drawbacks of conventional architectures based on a single processor: the overload imposed by operating system activities.

The architecture makes use of dedicated *hardware* units based on low cost microcontrollers. One microcontroller takes care of functions involving the management of task scheduling and time-dependent activation. Since scheduling algorithm tasks do not compete with application tasks anymore, they can be even more sophisticated and specialized. Another microcontroller is responsible for all activities related to inter-process communication, including the execution of communication drivers. It uses the processing capability to exchange data with the network, allowing the last microcontroller to expend more time in operations associated with the user's application. Not only the overall system performance is increased but the system behavior tends to be more deterministic, a very important characteristic when developing real-time applications.

1 INTRODUÇÃO

Um sistema computacional é considerado como "tempo-real" quando seu funcionamento depende não apenas do correto processamento lógico de entradas e saídas, mas também do exato instante de tempo em que sinais de saída são gerados e/ou sinais de entrada são amostrados. Assim sendo, uma das características principais de componentes usados em aplicações em tempo real é seu determinismo temporal. Ou seja, é imprescindível que estes componentes tenham a capacidade de responder a um estímulo, em qualquer circunstância, dentro de limites de tempo previamente determinados [Stankovic88] [HaSt91].

No passado recente, a grande maioria dos sistemas de tempo-real implementados eram estáticos por natureza. Os sistemas atuais estão tornando-se maiores e mais complexos e precisam operar em um sistema com requisitos dinâmicos e fisicamente distribuídos. Com estas características teremos os sistemas tempo-real da próxima geração caracterizados por uma arquitetura com controle altamente decentralizado, muitas vezes distribuído ao longo de uma área física de grandes dimensões.

Com os recentes progressos no campo da microeletrônica, eletrônica e computação, tornaram-se acessíveis microprocessadores com um excelente desempenho a custos extremamente baixos. A utilização deste tipo de componente possibilita o desenvolvimento de novas arquiteturas com características bastante interessantes para sistemas tempo-real distribuídos.

Seguindo esta tendência, este trabalho propõe a arquitetura de um *hardware* de baixo custo para desenvolvimento de aplicações em tempo real para sistemas de controle distribuído. O trabalho foi desenvolvido baseado na pesquisa e análise das arquiteturas convencionalmente desenvolvidas para este fim. Como conseqüência, chegou-se a uma arquitetura que visa eliminar pontos fracos detectados nas convencionais.

Esta dissertação apresenta no próximo capítulo um resumo de alguns conceitos importantes de sistemas tempo-real distribuídos, a fim de facilitar a compreensão do restante do trabalho.

No capítulo 3 é apresentado um estudo de como se encontram e como funcionam a maioria das arquiteturas de sistemas tempo-real distribuídos. Serão ressaltados os pontos identificados como limitantes de desempenho em tais arquiteturas.

No capítulo 4 é descrita a solução proposta, utilizando três processadores para eliminar os defeitos percebidos nas arquiteturas convencionais. São descritas as funções destinadas a cada um dos processadores presentes na arquitetura proposta, bem como a maneira com que foram implementadas características particulares de sistemas tempo-real distribuídos, como sincronização e troca de mensagens entre tarefas.

No capítulo 5 é descrito o *Hardware* implementado. São detalhadas as razões de engenharia que determinaram a escolha dos principais componentes utilizados, as mudanças de rumo tomadas e as melhorias que podem ser efetuadas.

No capítulo 6 são apresentados detalhes funcionais dos três micro-núcleos[2] que foram confeccionados para possibilitar a análise da eficiência da arquitetura desenvolvida. São comentadas todas as funções implementadas, as quais estão agrupadas com relação a que recurso físico ou de *software* estão relacionadas.

No capítulo 7 são detalhados os resultados obtidos com os testes desta arquitetura. São analisados e apresentados os tempos gastos em cada função extraída do microcontrolador principal, ressaltando o ganho de processamento atingido.

A conclusão possui uma abordagem de aspectos de custo/benefício da arquitetura apresentada, realçando as vantagens realmente obtidas, listando os próximos passos para dar continuidade ao trabalho e aproveitar o equipamento desenvolvido e sugerindo modificações que possam melhorar o resultado obtido.

No Apêndice A é relatada toda a sistemática de teste realizada para validação e avaliação da arquitetura desenvolvida. São apresentados os procedimentos de depuração do *software* e do *hardware*, e os requisitos de *hardware* previamente projetados foram importantes para alcançar um resultado satisfatório.

Por último o Apêndice B traz um guia de referência rápida, para aqueles que estiverem lendo esta dissertação e não estiver muito familiarizados com termos referentes a sistemas tempo-real distribuídos.

---

[2] Núcleo: Parte do sistema operacional multitarefa. Vide seção 2.4.

2 CONCEITOS DE SISTEMAS TEMPO-REAL DISTRIBUÍDOS

Existem muitas interpretações sobre o que é um sistema tempo-real, mas todas têm em comum o conceito apresentado por Stankovic em seu célebre artigo *"Misconceptions About Real-Time Computing"* publicado pela IEEE em 1988 e republicado em [StRa92]. Neste o autor salienta que a correção de sistemas tempo-real não depende apenas do resultado lógico da computação, mas também do tempo em que estes resultados são produzidos. Em [Motus93] é transcrita parte da norma DIN 44300 que estabelece:

" O funcionamento de um sistema tempo-real é uma operação de um sistema computacional no qual os programas para processar os dados de entrada estão constantemente ativos de forma que os resultados do processamento estarão disponíveis dentro de um pré-determinado intervalo de tempo."

É importante salientar que o fato de que estes requisitos temporais precisam ser atendidos, não implica necessariamente que o sistema seja "rápido" no sentido em que suas tarefas devam ser processadas em alguns milissegundos, como muitos são levados a pensar. A velocidade com que o sistema precisa responder está intimamente ligada com a aplicação a que este se destina. Podemos ter sistemas tempo-real que controlem um míssil teleguiado que precisa de respostas em milissegundos, como podemos ter sistemas, que também são classificados como de tempo-real, cujos requisitos temporais são da ordem de alguns segundos, como por exemplo o controle de algum processo térmico. O fundamental é que tendo sido fixado um intervalo de tempo máximo dentro do qual uma saída deva ser gerada em resposta a um estímulo surgido na entrada, este requisito temporal seja cumprido.

Isto fica claro, com a definição dada em [Young82]:

"Sistema tempo-real é aquele em que a atividade de processamento da informação deve responder a um estímulo de entrada externamente gerado dentro de um período finito especificado."

Seguindo estes conceitos, um sistema tempo-real é classificado em dois tipos principais:

- Sistema Tempo-Real Crítico ("Hard Real-Time")
- Sistema Tempo-Real Não Crítico ("Soft Real-Time")

Conforme definido em [BuWe90], sistema tempo-real crítico é aquele em que é absolutamente imperativo que a resposta ocorra dentro de um especificado período de tempo, caso contrário, coloca-se em risco a vida de seres humanos ou cifras consideráveis; sistema tempo-real não crítico é aquele em que o tempo de resposta é importante, mas se ocasionalmente um requisito temporal não for cumprido, o sistema continuará funcionando e as conseqüências não são desastrosas.

Pode-se recorrer a um exemplo de um manipulador de peças, Figura 2-1, com a função de retirar peças defeituosas de uma linha de produção e colocá-las em uma caixa de refugo para esclarecer o conceito de sistemas tempo-real, no caso um sistema tempo-real não-crítico. Suas tarefas são:

- Baixar o antebraço 90°.
- Fechar as garras.
- Levantar o antebraço 90°.
- Girar o tronco 90° para a direita.
- Abrir as garras.
- Girar o tronco 90° para a esquerda.

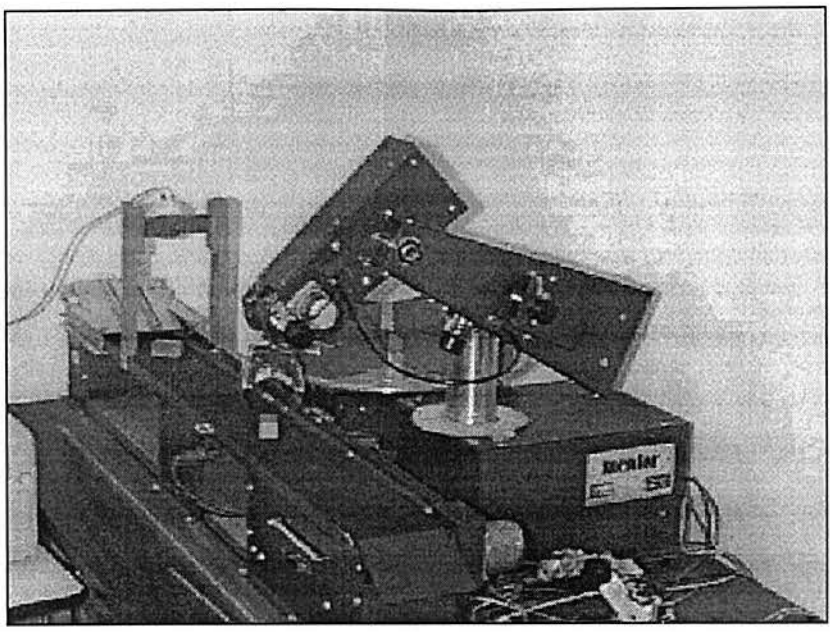


Figura 2-1 Manipulador de Peças

Considerando-se que todas estas tarefas foram executadas, podemos garantir que a peça defeituosa foi retirada da esteira e posta na caixa de refugo?

Não. Pois se o braço baixou, fechou as garras e levantou em um instante que não havia nenhuma peça sob seu braço na esteira ele não levantará nenhuma peça. Ou, então, se a peça que estiver sob seu braço não for a defeituosa, ele retirará uma peça, mas não a que deveria.

Para solucionar este problema, outra condição precisa ser imposta. Considerando a velocidade da esteira e o tamanho da peça, pode-se concluir que para o manipulador certamente pegar a peça, além de realizar todas suas tarefas, ele tem de fazê-las dentro de um período de tempo limitado, cujos valores dependem da dinâmica do processo. Pela necessidade deste requisito temporal para o sistema funcionar corretamente, fica evidenciado tratar-se de um sistema tempo-real.

Embora não se tenha uma uniformidade na comunidade científica no que tange as terminologias a serem empregadas em sistemas tempo-real distribuídos, este capítulo procurará descrever alguns conceitos importantes para tais sistemas. Os conceitos aqui descritos serão empregados no restante deste trabalho.

2.1 TAREFAS

Como relatado em [Tanenbaum92], normalmente nos computadores modernos temos a CPU (Unidade Central de Processamento) executando diversos programas simultaneamente. Enquanto um texto é digitado, por exemplo, dados podem estar sendo enviados à impressora ou lidos de um arquivo no disco. Nos sistemas multi-programas, a CPU alterna entre os diferentes programas sendo executados em frações de tempo de dezenas ou centenas de milissegundos. Embora instantaneamente somente um programa esteja sendo executado, como o chaveamento entre os diferentes programas ocorre muito rapidamente, o usuário tem a ilusão de um paralelismo, sendo por isto chamado de "quasi-paralelo". A utilização de um sistema que realmente execute atividades em paralelo é uma tarefa árdua. Desenvolvedores de sistemas operacionais têm se envolvido na definição de um modelo que facilite a implantação do paralelismo comentado.

Neste modelo, todo o programa executável no computador, freqüentemente incluindo também o sistema operacional, é organizado em tarefas. Uma tarefa é um programa executável, incluindo o valor corrente do contador de programa, registradores e variáveis. Conceitualmente, cada tarefa tem uma CPU virtual. De fato, só existe uma CPU cujo uso é "dividido" entre as tarefas. Para entendimento do sistema, entretanto, é muito mais fácil pensar neste como uma coleção de tarefas sendo executadas em paralelo. Este chaveamentos freqüentes entre diferentes tarefas faz com que o sistema seja definido como multi-tarefa.

Baseando-se na analogia relatada em [Tanenbaum92], fica mais claro a sutil, mas fundamental diferença entre um programa e uma tarefa. Têm-se uma mãe que se dispõe a preparar um bolo de aniversário para o seu filho. Nesta analogia, a receita é o programa (isto é, um algoritmo expresso em alguma notação que possa ser interpretado), a mãe é a CPU e os ingredientes são os dados de entrada. A tarefa consiste na atividade de ler a receita, misturar convenientemente os ingredientes e cozinhar o bolo. Agora imagine-se que entra na cozinha o filho chorando porque caiu e machucou seu joelho. Imediatamente a mãe parará a confecção do bolo, e irá procurar pelo livro de primeiros socorros para ver qual a providência correta a ser tomada neste caso. Após a criança ser medicada, ela retornará a confecção do bolo da etapa na receita em que havia parado. Neste caso percebemos que a "CPU" chaveou seu contexto da tarefa que estava realizando, confecção do bolo, para uma mais prioritária, atender o seu filho machucado.

A idéia principal é que uma tarefa é uma atividade que possui um programa, entrada de dados, saída de dados e um estado. Pela existência do estado da tarefa, esta pode ter seu processamento suspenso e posteriormente retomado sem que seja necessário recomeçar a tarefa. Um simples processador pode ser chaveado por diversas tarefas, sendo que o algoritmo usado para determinar quando deve parar o processamento de uma tarefa e passar a executar uma outra é denominado "algoritmo de escalonamento" (*scheduling algorithm*).

Em função dos evidentes benefícios que a programação multi-tarefa traz no atendimento a processos concorrentes, o processo de desenvolvimento de aplicações para sistemas tempo-real distribuídos envolve, normalmente, a decomposição destas em tarefas que serão responsáveis por partes do processamento necessário para a realização do trabalho.

Considera-se que cada tarefa pode ser executada de forma concorrente com as outras, facilitando a implementação da aplicação. A cada tarefa é atribuída uma prioridade. Quanto mais importante for uma tarefa, maior será a sua prioridade. Esta prioridade pode ser estática (fixa durante toda a existência da tarefa), ou dinâmica. Neste último caso, a prioridade pode ser modificada de acordo com conveniências decididas pelo algoritmo de escalonamento.

Uma tarefa pode estar em vários estados distintos. Estes podem ser apresentados como na Figura 2-2 [Ripps93].

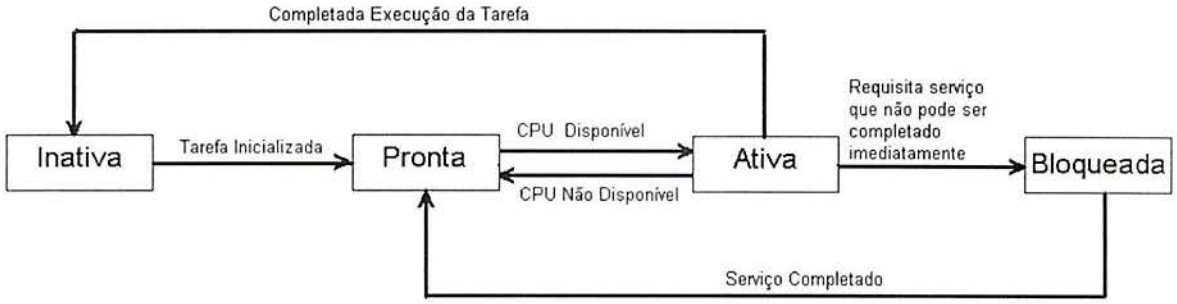


Figura 2-2 Estados possíveis de cada tarefa

- Ativa (ou Executando): A tarefa está em execução em um processador.

- Esperando Execução (ou Pronta para Execução): A tarefa está pronta para executar, porém o processador está ocupado por outra tarefa.

• Bloqueada: A tarefa aguarda um evento, tal como o término de um serviço requisitado, para continuar executando.

• Inativa: A tarefa já concluiu seu processamento ou ainda não foi requisitada para executar.

As possíveis transições entre esses quatro estados de uma tarefa são representados na Figura 2-2.

2.2 DETERMINISMO

Determinismo representa o grau de confiabilidade quanto ao seu sistema responder dentro de um determinado período de tempo [Gallmeister95]. Em sistemas tempo-real, deve ser possível garantir que a aplicação responderá dentro de um certo tempo, não importando quantas outras tarefas estejam rodando ou esperando pelo processador para serem executadas.

2.3 ALGORITMO DE ESCALONAMENTO

O algoritmo de escalonamento é utilizado pelo escalonador, que é a parte do núcleo responsável por determinar qual tarefa, entre as que estão prontas para executar, irá ter acesso ao processador. Os escalonadores normalmente baseiam-se nas prioridades de cada tarefa para tomarem suas decisões. Estas podem ser estáticas ou dinâmicas, como já explicado. Com esta filosofia, o acesso ao processador sempre será dado para a tarefa pronta para execução que estiver com a prioridade mais alta. O instante em que esta tarefa assumirá o controle do processador depende de qual classe de algoritmo de escalonamento está sendo utilizado.

• Algoritmo de Escalonamento Preemptivo: No momento em que uma tarefa pronta para execução possui prioridade mais alta do que a que está sendo executada pelo processador, o núcleo do sistema operacional substitui a tarefa que está sendo executada, salvando o seu contexto, colocando-a como pronta, e tornando a tarefa de prioridade mais alta ativa. Com um núcleo que possui um algoritmo preemptivo, a execução da tarefa com prioridade mais alta é determinística.

• Algoritmo de Escalonamento Não Preemptivo: Uma tarefa pronta para execução que possui prioridade mais alta do que a que está sendo executada, só será

processada quando a tarefa em processamento liberar, por algum motivo, o processador. Com um núcleo que possui algoritmo não preemptivo, diminui a necessidade de proteger variáveis compartilhadas (variáveis que podem ser acessadas por mais de uma tarefa). Esta não é uma regra absoluta; em alguns casos uma proteção com a utilização de ferramentas de *softwares* apropriadas pode ainda ser necessária.

Quanto à lógica a ser implementada como algoritmo de escalonamento de tarefas, existe uma variedade imensa. Estas buscam resolver uma dificuldade intrínseca em compatibilizar dois objetivos fundamentais [BuWe91]: garantir que os resultados serão produzidos no momento desejado e dotar o sistema de flexibilidade para adaptar-se a um ambiente dinâmico e, assim, aumentar sua utilidade.

Soluções de escalonamento que supõem um conjunto fixo de tarefas a serem executadas reservam recursos para o pior caso e são capazes de garantir que todas as tarefas serão concluídas no momento correto. Entretanto, aplicações construídas desta forma resultam em sistemas pouco flexíveis e na subutilização dos recursos computacionais [Oliveira97]. Mesmo com esta subutilização, estes algoritmos de escalonamentos muitas vezes são os escolhidos, em detrimento a algoritmos de escalonamentos mais flexíveis e "inteligentes". Isto deve-se ao fato de que estes últimos, embora teoricamente produzam melhores resultados, as vezes são tão complexos que o tempo gasto para que o mesmo seja processado não compensa o ganho de tempo obtido (cabe lembrar que, em geral, a execução do escalonador ocorre na mesma CPU que deve executar as tarefas da aplicação, caracterizando-se aí uma "disputa" pelo processador).

Dentre os algoritmos de escalonamento mais comumente utilizados, destacam-se:

- Prioridade Fixa. É o algoritmo mais largamente utilizado, principalmente devido a facilidade de implementação. Cada tarefa recebe uma prioridade quando da sua criação, a qual se mantém constante durante todo o ciclo de vida da tarefa. O escalonador decide qual a tarefa que deve ser executada, baseando-se na prioridade de cada uma daquelas que estão protas para executar. Escolhe sempre a de mais alta prioridade.

- FIFO. As tarefas vão sendo executadas na mesma ordem em que ficaram prontas para serem processadas.

• Taxa Monotônica *(Rate Monotonic* - RM). Proposto em [LiLa73], atribui prioridade mais alta para aquelas tarefas que possuem um período menor entre cada ativação. Trabalha com prioridade fixa.

• Tempo Limite para Terminar a Execução da Tarefa (*Deadline*) Mais Próximo Primeiro (*Earliest Deadline First* - EDF). Também proposto em [LiLa73], atribui uma prioridade maior para aquelas tarefas que possuem o *deadline* mais próximo do instante atual. Trabalha com prioridade variável.

• Menor Folga (*Least Slack).* Atribui uma prioridade maior para aquelas tarefas que possuem a menor folga para iniciarem o seu processamento e ainda conseguirem terminar de processar antes de atingir o tempo limite. Isto é, a folga é o período resultante da subtração do tempo que a tarefa necessita para ser processada, do tempo restante até a *deadline* ser atingida.

2.4 SISTEMA OPERACIONAL

2.4.1 Núcleo do Sistema Operacional (*Kernel*)

O núcleo é a parte do sistema operacional multitarefa responsável pelo gerenciamento das tarefas (gerenciamento do tempo de CPU) e comunicação entre tarefas. Um dos principais serviços realizado pelo núcleo é a troca de contexto [Labrosse92].

2.4.2 Troca de contexto

Quando o núcleo de um sistema operacional decide que a partir deste instante outra tarefa deve ser processada pelo processador, armazena o contexto da tarefa que está em execução em uma área específica da memória do sistema. Por contexto entende-se o conjunto de variáveis que serão necessárias para que a tarefa possa posteriormente retomar seu processamento como se este não tivesse sido interrompido. Este conjunto de variáveis inclui os registradores, o contador de programa, o ponteiro da pilha, entre outros.

2.5 COORDENAÇÃO E SINCRONIZAÇÃO

Embora uma tarefa seja fisicamente completa e executável, ela não é logicamente independente das outras. Cada tarefa realiza uma pequena parte de uma aplicação total, trabalhando em paralelo com outras tarefas. Em vários pontos, as tarefas devem coordenar

suas atividades. Por exemplo, uma tarefa que realiza a secagem da tinta de um automóvel em uma montadora automatizada deve esperar que a tarefa responsável pela pintura termine seu trabalho. Esta, por sua vez, só pode começar seu trabalho após a tarefa responsável pela colocação da porta no lugar onde ela deve ser pintada tenha terminado seu trabalho. Similarmente, quando várias tarefas emitem relatórios em um terminal compartilhado, cada uma delas deve aguardar até obter acesso exclusivo ao recurso compartilhado. De outra forma, os relatórios resultantes terão suas aproveitabilidades comprometidas.

Portanto, tarefas não são totalmente independentes. Elas compartilham objetivos, dados e recursos. Como resultado, elas necessitam do sistema operacional para coordenar suas execuções individuais. O planejamento da forma como as tarefas se coordenarão e comunicarão é uma das principais partes do projeto de uma aplicação em tempo real [Ripps93].

Sincronização é o bloqueio de uma tarefa até que alguma condição especificada seja encontrada. De acordo com essa definição, uma tarefa pode se sincronizar com eventos físicos, como também consigo mesma e outras tarefas. Por exemplo, quando uma tarefa sofre uma pausa e é bloqueada durante um determinado tempo preestabelecido, a tarefa está sincronizada com o relógio físico.

Pode-se definir comunicação como a transferência de informações entre tarefas. A comunicação pode ocorrer diretamente de uma tarefa para outra. Por exemplo, quando uma tarefa programa a ativação de uma outra tarefa, esta pode receber um argumento da primeira. Freqüentemente, contudo, a comunicação utiliza um objeto separado, chamado trocador de mensagens - uma fila pública, onde qualquer tarefa pode enviar ou receber uma mensagem.

A comunicação entre tarefas utiliza diferentes artifícios para serem efetivadas. Os meios mais comuns de realizá-las é através de troca de mensagens, grupo de *flags* de eventos, semáforos e variáveis compartilhadas controladas.

As trocas de mensagens facilitam a comunicação entre tarefas. Normalmente, a comunicação entre tarefas envolve o transporte de informação, tal como um conjunto de parâmetros para algum trabalho a ser feito ou o estado de um trabalho que está sendo realizado em estágios por uma seqüência de tarefas. O acoplamento também envolve coordenação. Para um receptor, isso significa a aptidão para esperar pela chegada de uma mensagem. Para o remetente, significa a aptidão para esperar por uma mensagem a ser

enviada. Portanto, mesmo que não haja conteúdo na mensagem, a passagem de mensagens pode ainda coordenar as atividades das tarefas.

Um *flag* de evento pode também acoplar tarefas, mas com as informações restritas a um único bit. Comparativamente, uma mensagem sem conteúdo também carrega um único bit de informação que está presente em um trocador ou não. Mesmo assim, a coordenação através de flags de evento e mensagens é essencialmente diferente. Quando um *flag* de evento está ligado, todas as tarefas estão esperando que o evento continue e o *flag* de evento permanece ligado para quaisquer tarefas que venham mais tarde examiná-lo. Com uma mensagem, a tarefa que a recebe, consome-a; caso existam duas tarefas esperando por uma mensagem, somente uma continua enquanto a outra continua aguardando [Ripps93].

3 ARQUITETURAS CONVENCIONAIS

Sistemas tempo-real distribuídos consistem de uma rede de Unidades de Processamento (UP) autônomas, interligadas entre si e com sensores e atuadores. Um protocolo de comunicação é utilizado para interconectar os diversos elementos de rede de modo que se possa controlar uma planta, como mostrado na Figura 3-1.

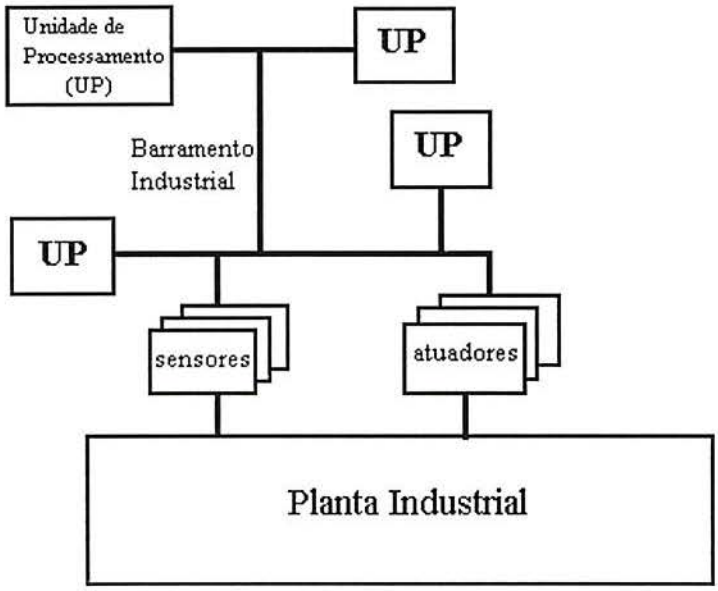


Figura 3-1 Sistema Tempo-Real Distribuído

Cada UP é responsável pelo processamento de um conjunto de processos cooperativos seqüenciais: as tarefas. As implementações convencionais de sistemas operacionais distribuídos para aplicações de tempo-real, incluem funções de *software* para gerenciar a utilização dos recursos de *hardware* disponíveis. Estas geralmente utilizam um relógio (*timer*) para auxiliar na ativação de tarefas com requisitos temporais. Tal configuração possui diversas desvantagens, em particular, as seguintes:

- **Escalonamento intrusivo de tarefas:** nas implementações convencionais de sistemas operacionais distribuídos, o algoritmo de escalonamento, geralmente, é executado no mesmo processador que as tarefas da aplicação. Devido a esta característica, algoritmos mais elaborados, que tenderiam a uma alocação ótima dos recursos do processador disponível, não

são utilizados por dispenderem muito tempo de processamento nas suas próprias rotinas. Como conseqüência, temos, na prática, o uso de algoritmos simples que levam a uma solução não ótima, como por exemplo Prioridade Fixa.

• **Gerenciamento inadequado de tarefas com requisitos assíncronos:** Existem muitas situações em sistemas de tempo-real em que ações devem ser tomadas como respostas a eventos assíncronos (por exemplo: "caso temperatura ultrapassar *valor-limite*" ....). Duas abordagens são comumente utilizadas por sistemas operacionais convencionais: i) pela geração de uma interrupção; ii) por uma amostragem periódica (*polling*). Ambos ocasionam uma sobrecarga considerável ao processador, devido a chaveamentos de contexto desnecessários, uma vez que os procedimentos de verificação de ocorrência de eventos bloqueiam as tarefas que estão sendo executadas. Com isto, uma considerável quantidade de tempo de processamento é desperdiçado.

No caso do *polling*, existe uma desvantagem importante: visando reduzir a latência do sistema, ou seja, o intervalo de tempo entre a ocorrência de um evento assíncrono e a reação do sistema, busca-se adotar um período de *polling* tão pequeno quanto possível. Por outro lado, este aumento na freqüência de *polling* aumenta consideravelmente a sobrecarga no processador, uma vez que na maior parte das leituras nada ocorrerá. Estabelece-se, então, uma situação onde busca-se uma relação custo/benefício ótima quanto a este compromisso. No caso de ser gerada uma interrupção na chegada de um evento assíncrono, este compromisso não aparece. Normalmente, esta é a opção para sistemas com poucos eventos assíncronos; no entanto, quando o número de eventos assíncronos começa a crescer, esta opção começa a perder atratividade devido a elevada complexidade que o *hardware* pode alcançar.

• **Gerenciamento intrusivo do instante de ativação das tarefas com requisitos temporais:** sistemas de tempo-real necessitam executar ações específicas em períodos de tempo pré-determinados. Tarefas com requisitos temporais são chamadas temporizadas. Cada tarefa síncrona pode ser classificada, quanto ao seu instante de ativação, em três tipos:

•Cíclica. Ex: A cada 5 segundos deve ser atualizado o display.

•Absoluta. Ex: Às 12 horas deve soar a sirene.

•Relativa. Ex: Após 10 segundos do acionamento da esteira, o pistão deve ser acionado.

Para gerenciamento destas tarefas o sistema operacional baseia-se em um relógio interno, o qual é atualizado pela ativação de um sinal de interrupção periódico. A cada ativação, a rotina de atendimento à interrupção do relógio atualiza as estruturas de dados relativas ao tempo de ativação apropriadas e verifica, na maioria das vezes sem sucesso, se alguma das tarefas com requisitos de ativação temporal teve seu instante de ativação atingido. A freqüência com que será programada a interrupção do relógio é uma relação de custo/benefício que precisa ser ponderada, pelos mesmos motivos apresentados no gerenciamento de tarefas com requisitos assíncronos que utilizam *polling* periódico.

• **Influência das funções de gerência das informações destinadas ou oriundas da rede de comunicação no desempenho da UP:** as funções de gerência das informações oriundas da rede de comunicação a que a Unidade de Processamento está conectada e a adequação de mensagens desta UP para outras, e vice-versa, também retiram tempo de processamento das tarefas da aplicação por parte do processador.

Como resultado destes problemas relatados, chega-se a uma Unidade de Processamento que tem sua capacidade de execução das tarefas da aplicação diminuída, já que os recursos de processamento disponíveis precisam ser compartilhados entre estas e funções que permitem a operacionalização da UP. Conforme será visto na seção 7, dados obtidos em uma aplicação real mostram que esta sobrecarga pode ser superior a 25%.

4 ARQUITETURA PROPOSTA

Pela análise dos problemas das arquiteturas convencionais, explicadas no capítulo anterior, pode-se agrupar o conjunto de funções que o microprocessador tem de desempenhar em três grupos distintos:

- Funções destinadas a possibilitar a troca de mensagens entre tarefas, quer sejam elas pertencentes à mesma unidade de processamento (UP) ou não;

- Funções destinadas ao gerenciamento das tarefas da aplicação, a fim de possibilitar que estas sejam processadas tão logo quanto possível, após terem suas condições temporais ou eventuais alcançadas;

- Tarefas da aplicação;

Destas atividades, apenas a última resulta na efetiva execução das tarefas a serem desempenhadas pela aplicação, sendo que as outras são tarefas puramente administrativas e que, por isto, devem ser minimizadas. Numa primeira tentativa de aliviar-se a carga imposta ao sistema operacional tempo-real distribuído, apresentou-se em [HWP95] a proposta de um gerenciador de interrupções e eventos assíncronos, onde busca-se, com o uso de um componente especial, controlar todas as funções que possuem/necessitam requisitos temporais, evitando desnecessários chaveamentos de contexto do microcontrolador. Estes chaveamentos levam a um decréscimo do poder de processamento da UP.

Partindo desta proposta inicial, o presente trabalho busca incluir outras funções administrativas, como o algoritmo de escalonamento e a gerência de comunicação entre UPs, como funções a serem retiradas do processador das tarefas da aplicação. Outro objetivo é utilizar componentes eletrônicos "de prateleira", disponíveis a baixo custo, ao invés de utilizar componentes desenvolvidos especialmente para o suporte de *hardware* planejado. Para isto lançar-se-á mão de microcontroladores existentes no mercado de circuitos integrados comerciais. A motivação para esta decisão advém, também, do fato de que estes componentes têm poder de processamento e alguns periféricos embutidos interessantes.

Utilizar-se-á um microcontrolador para cada um dos três grupos de funções detectados e citados anteriormente. Com isto, boa parte das funções que a UP é obrigada a realizar, para que as tarefas da aplicação sejam processadas, serão feitas por microprocessadores diferentes daquele responsável pelo real processamento das tarefas da aplicação. Como comentado no capítulo referente aos resultados obtidos, a necessidade de se separar o bloco Administrador do bloco de Comunicação, estará intimamente ligada a carga de processamento que a rede chão-de-fábrica a qual a UP estará conectada solicitar para o atendimento de seu protocolo. Este trabalho buscou por separar estas atividades em processadores diferentes permitir a arquitetura uma adequação a um maior número de casos.

Os microcontroladores dedicados a cada um destes grupos serão referenciados no transcorrer desta dissertação como a seguir se apresenta:

- Do bloco Executor: **Microcontrolador Principal (μPrincipal)**
- Do bloco Administrador: **Microcontrolador Gerente (μGerente)**
- Do bloco Comunicação: **Microcontrolador de Comunicação (μComunicação)**

A Figura 4-1 apresenta um diagrama esquemático da arquitetura proposta.

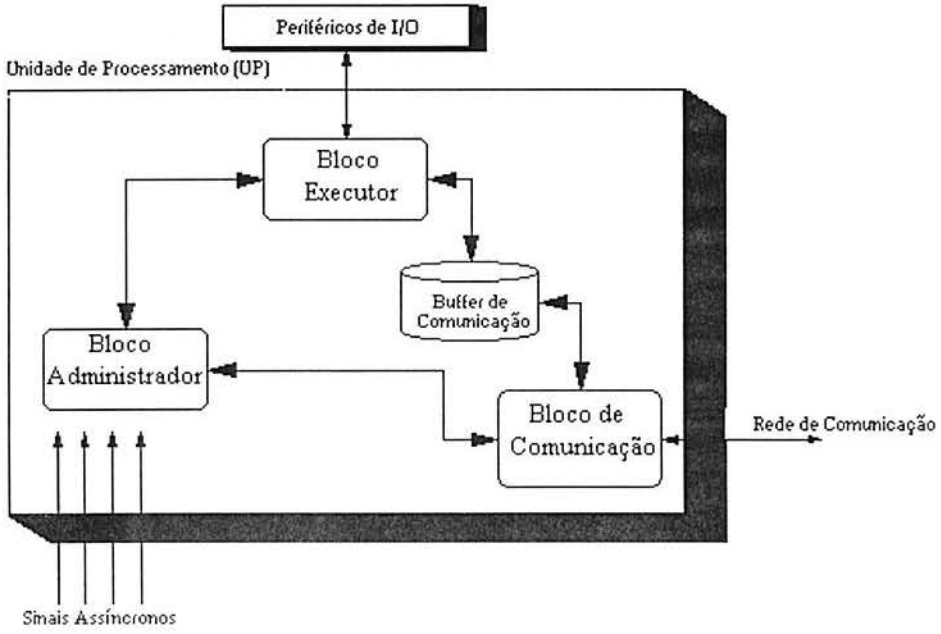


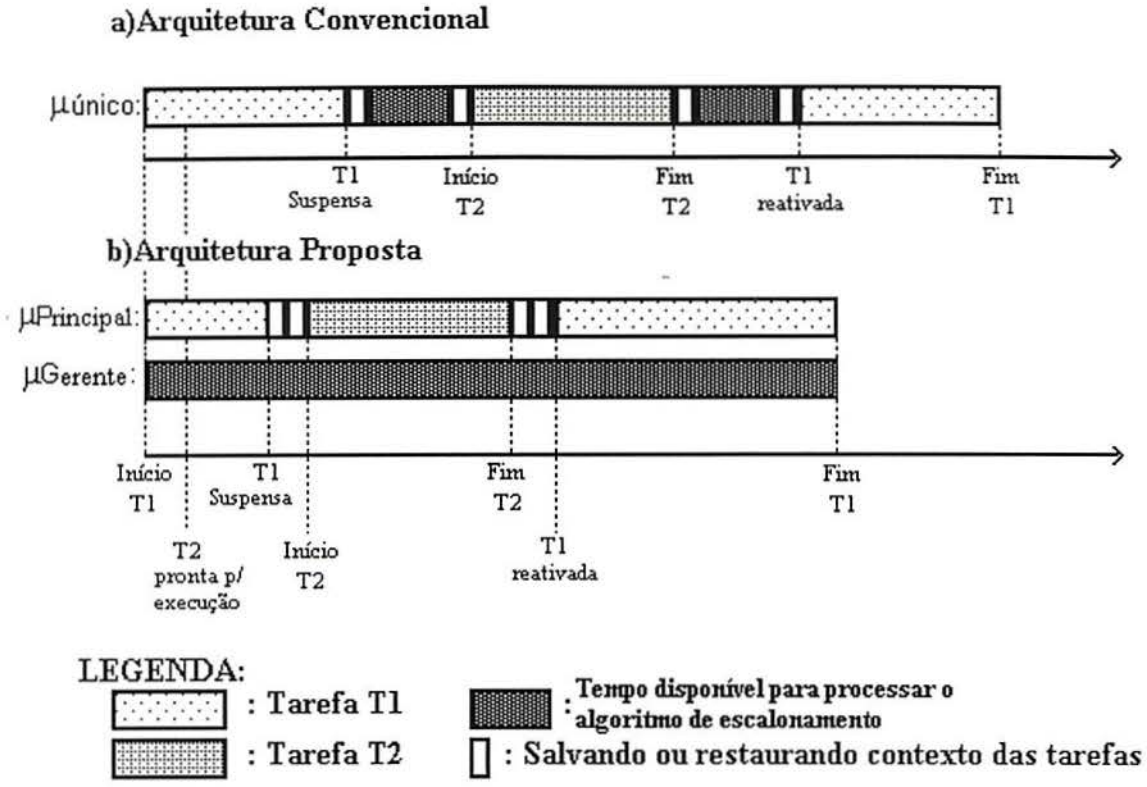
Figura 4-1 Unidade de Processamento (UP)

4.1 BLOCO ADMINISTRADOR

O Bloco Administrador é o responsável pelas funções de escalonamento das tarefas, bem como pela gerência de todos os eventos síncronos ou assíncronos. Entende-se por eventos síncronos, todos aqueles que dependem do Relógio do sistema para terem seus instantes de ativação atingidos, quer seja esta dependência periódica, absoluta ou relativa.

O chamado microcontrolador Gerente (µGerente), integrante do bloco Administrador, é o responsável por determinar qual tarefa estará sendo executada pelo µPrincipal. Nenhuma tarefa é processada sem o conhecimento do µGerente. Suas principais atividades são:

- **Controlar o escalonamento de tarefas.** É do µGerente a responsabilidade de determinar a ordem em que as tarefas prontas para execução deverão ser processadas. Também é responsabilidade do µGerente, relacionar cada tarefa esperando o momento em que ficarão prontas para execução com o respectivo evento que estão a espera, quer seja ele um requisito temporal, um sinal assíncrono externo, ou a chegada de uma mensagem. É importante salientar que com o algoritmo de escalonamento agora sendo processado por um microcontrolador diferente do que processa as tarefas da aplicação, o desempenho do sistema é naturalmente incrementado, como mostrado na Figura 4-2. Uma vez que o algoritmo de escalonamento roda em um processador distinto, este pode ainda ser mais elaborado, visando aumentar a qualidade do escalonamento gerado, pois a sobrecarga excessiva que apresentaria em uma arquitetura convencional é agora minimizada.



OBS: Exemplo em que durante a execução de T1, T2, que é mais prioritária, fica pronta para execução.

Figura 4-2 Influência do algoritmo de escalonamento em cada arquitetura

- **Gerenciar qual tarefa estará sendo executada pelo bloco Executor a cada instante.** O µGerente possui sempre o controle de qual tarefa está sendo executada em cada instante pelo bloco Executor, sendo que um chaveamento de tarefas somente ocorrerá se:

  - A tarefa sendo executada for finalizada. Neste caso, o µPrincipal informa ao µGerente este fato. Imediatamente, o µGerente responde informando dentre as tarefas que se encontram na fila de tarefas prontas para execução, qual a mais prioritária, portanto, a próxima a ser executada.

  - A tarefa em execução precisa ser suspensa até que determinado evento ocorra. O µGerente age de forma semelhante ao primeiro modo, acrescentando-se que a tarefa suspensa será relacionada ao evento (há uma lista de tarefas suspensas). Quando este evento ocorrer, a tarefa suspensa retorna à fila de tarefas prontas para execução. Este evento pode ser o transcorrer de um determinado tempo, a chegada de um sinal assíncrono, a chegada de uma mensagem para a tarefa suspensa ou uma combinação destes tipos de eventos.

- O µGerente detecta que uma tarefa pronta para executar possui uma prioridade maior que a atualmente em execução no µPrincipal. Neste caso, o µGerente solicita ao µPrincipal que interrompa o atual processamento e comece a executar a nova tarefa.

- **Controlar a ativação de tarefas síncronas.** Tendo recebido do µPrincipal um Pedido de Ativação de Tarefa (PAT) [PLH96], este é interpretado, podendo ser tanto dependente de um relógio absoluto (tarefas síncronas) quanto de um evento externo (tarefas assíncronas). Caso seja detectado tratar-se de tarefa síncrona, é extraída a informação da hora em que esta deve ser ativada.

Esta informação pode ser absoluta, relativa ou cíclica, conforme exemplificado na seção anterior. Cada um dos tipos recebe tratamento adequado para que o próximo instante de ativação seja expresso em um horário absoluto. Este horário é, então, colocado em uma fila chamada SIFO (*Smallest Input First Output*) [HWP95]. O funcionamento da memória SIFO é semelhante ao de uma memória FIFO, com exceção de que além de ser um *buffer* de dados, esta ordena os dados inseridos em ordem crescente do instante de ativação das tarefas.

Com este tipo de ordenação, basta ao µGerente se preocupar com o instante de ativação do primeiro elemento da SIFO, pois os outros instantes de ativação só serão alcançados posteriormente àquele que ocupa a primeira posição.

Para liberar o µGerente de consultas desnecessárias sobre a chegada ou não do instante de ativação de uma tarefa programada na SIFO, foi incluído no bloco Administrador um Relógio de Tempo-Real (RTC). A este é enviado o instante de ativação da próxima tarefa. O RTC, então, fica programado para enviar ao µGerente uma interrupção assim que detectar a chegada deste instante.

Com a chegada da interrupção, o µGerente verifica a prioridade da tarefa relacionada a este instante com a da tarefa em execução pelo µPrincipal. Caso a tarefa cujo tempo de ativação recém foi alcançado tiver menor ou igual prioridade, ela é posta na fila de tarefas prontas, que são ordenadas por prioridades. Caso contrário, o µPrincipal recebe um aviso para suspender o processamento da tarefa que está atualmente processando, e começar a processar esta nova tarefa. Por sua vez, o µGerente coloca a tarefa que teve seu processamento interrompido na fila de tarefas prontas, possibilitando que seu processamento seja retomado

posteriormente. Na seqüência, programa o RTC com o instante de ativação da próxima tarefa da SIFO.

• **Controlar a reativação das tarefas cíclicas.** Quando a execução de uma tarefa cíclica é finalizada, o µGerente tem de reprogramar o instante de ativação desta tarefa, acrescentando-o na SIFO.

• **Monitorar eventos assíncronos.** O µGerente é responsável pela verificação periódica da ocorrência de eventos assíncronos nas entradas disponibilizadas para este tipo de eventos pela arquitetura. Quando alguma das entradas estiver ativada, o µGerente verifica quais as tarefas que dependem da ativação deste evento. Com aquelas tarefas que só esperavam este evento para se tornarem ativas, o µGerente procede semelhantemente ao adotado com as tarefas síncronas que tornaram-se ativas. Como as tarefas processadas pelo µGerente não competem com as tarefas de aplicação, não existe o problema de custo/benefício abordado na análise das arquiteturas convencionais. Portanto, o período de *polling* pode ser tão curto quanto necessário.

• **Armazenar a prioridade da tarefa sendo executada.** Uma tarefa sendo executada pelo µPrincipal só sofrerá interrupção no seu processamento quando realmente existir uma tarefa de maior prioridade pronta para ser processada. Para que isto seja possível, o µGerente mantém, junto com o identificador da tarefa sendo processada, a prioridade da mesma. Comparando estas prioridades, desnecessários salvamentos de contexto, como mostrado na Figura 4-3, podem ser evitados. Cabe ressaltar que algoritmos de escalonamento que consideram prioridade como função do tempo (ex: "*Least Slack*") encontram nesta arquitetura solução para o problema de sobrecarga excessiva no processador que surgia com o seu uso em arquiteturas convencionais. Nesta nova arquitetura pode-se constantemente estar recalculando a prioridade das tarefas sem perda de desempenho.

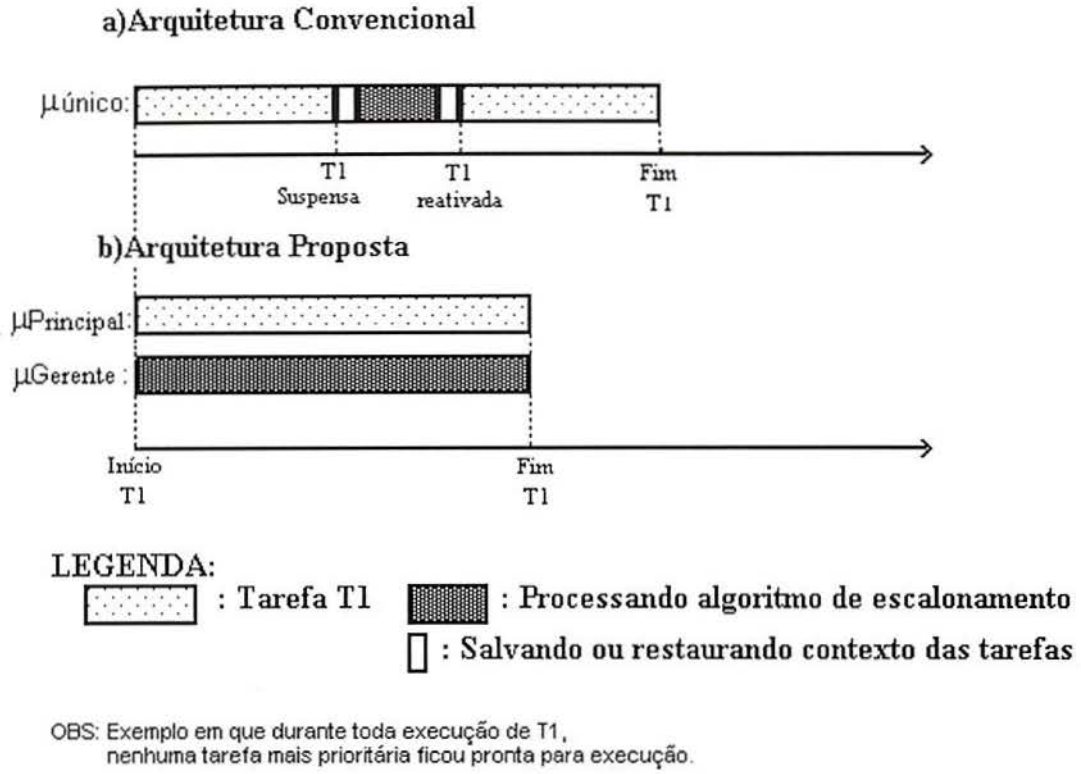


Figura 4-3 Eliminação de chaveamento de contexto desnecessário

• **Guardar estatísticas sobre o tempo médio de execução de cada tarefa.** A cada troca de contexto, o µGerente atualiza as informações de tempo de execução das tarefas envolvidas. Este tipo de informação poderá ser de grande utilidade na implementação de algoritmos de escalonamento mais "inteligentes", os quais podem levar em conta o tempo de execução médio de cada tarefa. Além disso, estas estatísticas fornecem valiosas informações "temporizadas" para avaliação/documentação.

4.2 BLOCO EXECUTOR

O Bloco Executor basicamente é o responsável pelo processamento das tarefas da aplicação. Este possui como seu componente fundamental o Microcontrolador Principal (µPrincipal). Suas principais funções são:

• **Executar as tarefas da aplicação.** É o Bloco executor que realmente processa as tarefas da aplicação. Isto é, processa as tarefas programadas pelo usuário. Estas tarefas são as que deram razão ao desenvolvimento e implantação do sistema pelo usuário.

• **Gerenciar a área de memória associada a cada tarefa da aplicação.** A cada tarefa da aplicação é associada uma área de memória independente. Nesta área ficam armazenadas informações para controle de fluxo de execução de cada tarefa, como: endereço da próxima instrução que deve ser executada desta tarefa; tamanho e endereço da pilha particular desta tarefa; tipo de salvamento de contexto que deve ser efetuado e outras informações que sejam necessárias.

• **Efetuar o salvamento e restauração de contexto das tarefas da aplicação.** Sempre que a tarefa sendo executada tiver seu processamento suspenso, quer seja pela necessidade de espera de algum evento para continuar seu processamento, quer seja pelo recebimento de um aviso do µGerente de que uma tarefa mais prioritária necessita ocupar o µPrincipal, será realizado um salvamento do contexto da tarefa sendo processada.

• **Trocar mensagens entre tarefas.** A troca de mensagens entre tarefas é realizada de maneira simplificada, sendo transparente para o µPrincipal o fato de a tarefa destino localizar-se dentro de sua UP ou em outra. O Bloco Executor simplesmente coloca a mensagem no *buffer* de comunicação e deixa para o Bloco de Comunicação a responsabilidade de formatar a mensagem segundo a necessidade da tarefa destino.

4.3 BLOCO DE COMUNICAÇÃO

O Bloco de Comunicação é responsável por qualquer tipo de comunicação entre tarefas, pela comunicação com outras Unidades de Processamento e pela comunicação com outros dispositivos de *hardware*, tais como sensores inteligentes e atuadores. Suas principais funções são:

• **Gerenciar comunicação entre tarefas da mesma UP.** Quando recebe informação de que existe uma mensagem disponível para ser transmitida no *Buffer* de Comunicação, cujo destino localiza-se na mesma UP, toma as seguintes providências: repõe a mensagem no *Buffer* de Comunicação; avisa µGerente que existe uma mensagem disponível para uma determinada tarefa, acrescentando quem enviou e qual tamanho da mensagem.

• **Gerenciar comunicação de tarefas da sua UP para tarefas de outra UP.** No caso da mensagem a ser enviada ter como destino outra UP, toma as seguintes providências: acrescenta à mensagem os bytes de controle necessários para que a mesma fique no formato

exigido pelo protocolo de comunicação da rede que está sendo utilizado (Profibus, Fieldbus, etc.); envia a mensagem para a rede de comunicação, assim que esta lhe permitir.

- **Gerenciar comunicação de tarefas de outra UP para tarefas da sua UP.** Quando perceber na rede de comunicação uma mensagem cujo destino é a sua própria UP, capta esta mensagem, tomando as seguintes providências: Extrair da mensagem os bytes de controle e adequando-a ao formato simplificado interno de troca de mensagens; avisa µGerente que existe uma mensagem disponível para uma determinada tarefa, acrescentando quem enviou e qual tamanho da mensagem.

- **Possibilita a sincronização entre sua UP e as demais.** Quando recebe solicitação do horário de sua UP por meio da rede de comunicação, toma as seguintes providências: dispara *timers* internos para que a resposta seja a mais precisa possível, isto é, que possa informar a hora que o relógio de sua UP marcava quando a solicitação foi realizada; solicita ao µGerente a hora atual do relógio da UP; responde, via rede de comunicação, ao questionador o horário de sua UP.

5 PROJETO DE HARDWARE

Visando possibilitar a validação da arquitetura proposta, foi projetado um *hardware* que implementasse esta arquitetura. Neste capítulo será apresentado o *hardware* desenvolvido, Fotografia (Figura 5-1) e Diagrama de Blocos (Figura 5-2) a seguir, detalhando as razões de engenharia que conduziram às decisões tomadas.

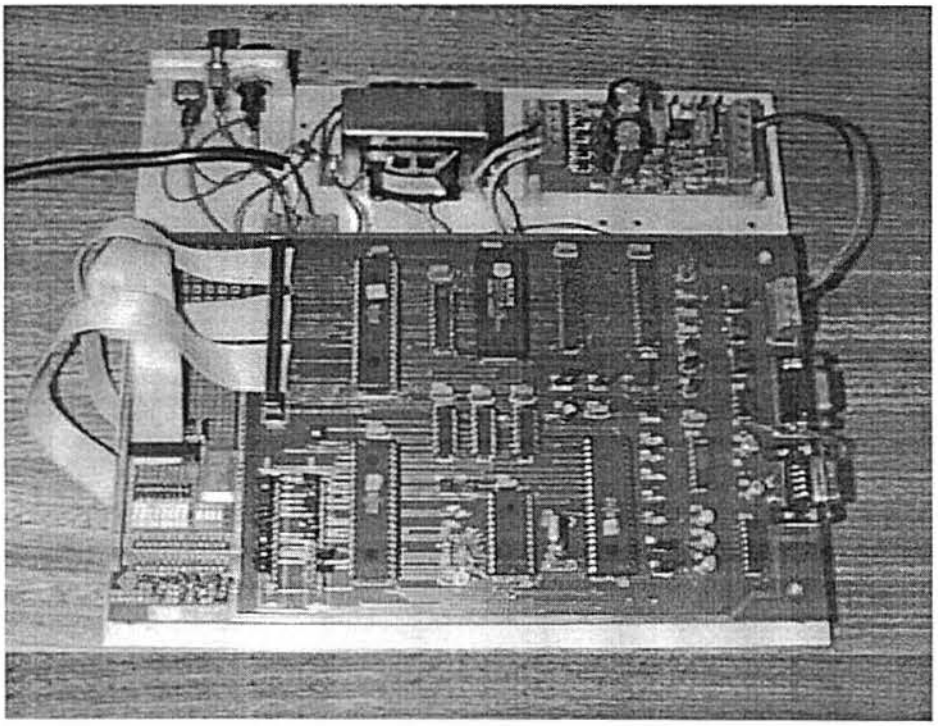


Figura 5-1 Fotografia do Hardware desenvolvido conforme nova arquitetura proposta

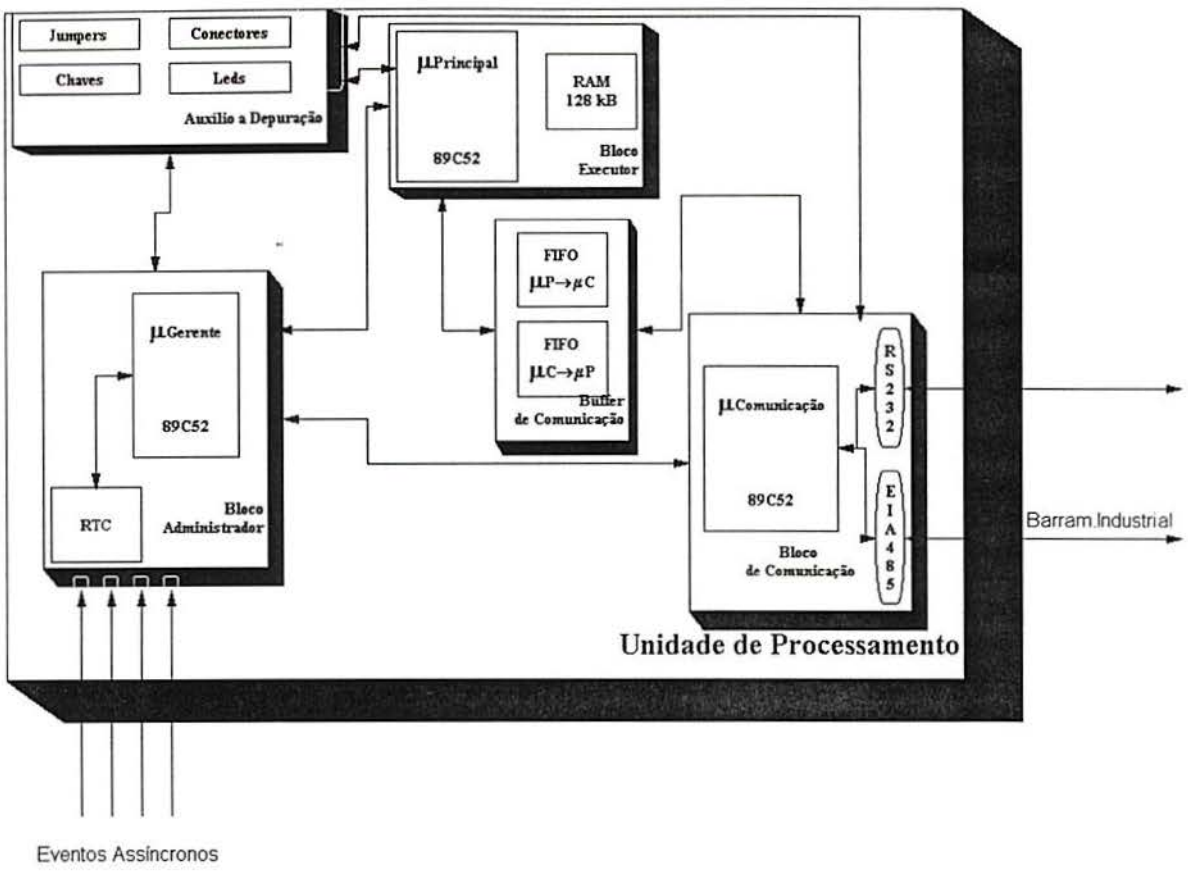


Figura 5-2 Diagrama de blocos do *Hardware* implementado

5.1 MICROCONTROLADORES UTILIZADOS

A primeira tomada de decisão importante que se fez necessária foi quanto a que família de microcontroladores utilizar. A preferência recaiu sobre a família '51, por dois motivos principais:

• Como pretende-se que o resultado desta dissertação seja utilizado para pesquisa e ensino de Sistemas Tempo-Real para a graduação e pós-graduação em Engenharia Elétrica, optou-se pelo uso da família mais popularizada e disseminada entre os Engenheiros Eletricistas em formação ou já formados por esta Universidade, bem como de ampla utilização pelas indústrias do setor Eletro-Eletrônico local. Cabe ressaltar que a disciplina de Microprocessadores-I deste Departamento utiliza exatamente esta família de microcontroladores como base teórica e prática de suas aulas. Assim procedendo, ficaria

facilitado o uso do *hardware* desenvolvido com esta arquitetura aqui apresentada por quem desejar se aprofundar no tema de sistemas tempo-real distribuídos.

• O outro importante fator que fez com que se decidisse pela utilização da referida família de microcontroladores foi o fato de possuir o Departamento o "C51 Compiler" da Keil *Software* Inc., ferramenta de suporte computacional para programação, compilação e depuração de sistemas usando "C", facilitando o trabalho de quem desejar desenvolver aplicações com estes componentes. Outro *software* importante existente no mesmo pacote é o Dscope 251/51 for Windows que possibilita simulação da aplicação desenvolvida, permitindo que a depuração do *software* escrito seja muito mais simplificada. Com os *softwares* adquiridos, a larga variedade desta família de microcontroladores pode ser explorada facilmente pois os programas já vem com bibliotecas de diversos dos integrantes desta família. Isto faz com que a migração para microcontroladores que para uma determinada aplicação sejam mais apropriados possa ser feita sem maiores contratempos.

Tendo sido determinado com qual família de microcontroladores se iria trabalhar, o passo seguinte foi a escolha de qual dos componentes desta família seria o mais apropriado para a arquitetura proposta. Decidiu-se que para o desenvolvimento do protótipo da aplicação, o mais apropriado seria a utilização dos microcontroladores da Atmel Corporation. Os principais fatores que conduziram a esta escolha foram:

• Os microcontroladores desenvolvidos por esta empresa caracterizam-se por terem, internamente ao chip, memórias não-voláteis do tipo *Flash EEPROM* em substituição aos microcontroladores tradicionais que utilizam como tais memórias as do tipo ROM ("Read Only Memory") PROM ("Programable Read Only Memory") ou EPROM ("Eraseble Read Only Memory"). A grande vantagem deste tipo de memória para o desenvolvimento de protótipos é que a etapa de depuração do *software* integrado com o *hardware* fica extremamente facilitada. Isto deve-se ao fato de que o apagar do conteúdo da memória do componente é feito eletricamente, em poucas unidades de segundos, enquanto os componentes com EPROM levam normalmente mais de dez minutos e os componentes com PROM E ROM não podem ter o conteúdo da memória apagado.

• A necessidade de um gravador/apagador deste tipo de componente era emergente para o laboratório, visto que diversos outros projetos já os estavam utilizando. Por termos disponível no *web site* da empresa o projeto eletrônico e o *software* para o

gravador/apagador destes componentes disponível, a construção deste equipamento era um fator de fácil implementação.

• A aquisição destes componentes é um outro aspecto de fácil solução. Diferentemente de componentes de outras empresas que ou não possuem representantes no Brasil, ou quando os possuem, estes só aceitam encomendas em grande quantidade, os componentes da Atmel Corporation podem facilmente serem adquiridos em um dos seus representantes em São Paulo[3], e em quantidades pequenas.

Entre os microcontroladores fabricados pela Atmel, optou-se pelo AT89C52 como o que seria utilizado durante o desenvolvimento da arquitetura e testes necessários para validação da mesma. Os principais fatores foram:

• Ser um componente de custo relativamente baixo.

• Ser um componente trivial, portanto, normalmente disponível para pronta entrega pelo fornecedor. Fator importante, principalmente quando se deseja adquirir pouca quantidade.

• A existência, nos *softwares* adquiridos pelo departamento como ambientes de desenvolvimento de microcontroladores da família '51, do *driver* pronto para tal componente.

• Possuir algumas vantagens comparativas interessantes quando comparado ao outro microcontrolador da linha da Atmel Corporation, AT89C51, que também preencheria os três primeiros itens aqui listados, quais sejam:

	AT89C51	**AT89C52**
Flash (Bytes)	4k	8k
Ram (Bytes)	128	256
Temporizadores/Contadores	2	3
Fontes de Interrupção	6	8

Destes itens, é possível destacar os de maior importância: o tamanho da Ram e o número de Temporizadores. O tamanho da Ram é um fator crítico, principalmente para o microcontrolador que desempenhará o papel de µGerente. A quantidade de Ram disponível é

diretamente proporcional à quantidade de tarefas simultâneas que podem ser programadas. Não fosse os motivos já relatados para a escolha de um microcontrolador da linha da Atmel para o desenvolvimento do *hardware*, provavelmente seriam escolhidos alguns que se caracterizam por possuírem uma capacidade de Ram interna ainda maior. A idéia é que, após a estabilização do *software* que irá desempenhar a função de µGerente, este seja substituído por outro que possua 1 KByte de Ram interna, como, por exemplo, alguns microcontroladores da Philips que já possuem esta característica. Com isto, teremos aumentada a capacidade de tarefas que o Bloco Administrador pode trabalhar simultaneamente.

5.2 BLOCO ADMINISTRADOR

O principal componente do Bloco Administrador é o µGerente, implementado com o microcontrolador 89C52. As funções atribuídas a cada pino estão detalhadas a seguir:

	NOME RECEBIDO	FUNÇÃO
Porta 0	MGD0..MGD7	Utilizado como barramento de dados para o RTC.
Porta 1	INT0..INT7	Oito entradas de eventos assíncronos. Estas são acessadas por *polling* pelo núcleo do sistema operacional sendo executado neste componente.
Porta 2 bit0 a bit4	MGRTC0..MGRTC4	Utilizados para informar qual comando ou função que deve ser realizado pelo RTC.
Porta 2 bit5	MGRTCRD#	Pino responsável pela ativação da leitura das informações requisitadas ao RTC.
Porta 2 bit6	MGRTCWR#	Pino responsável pela ativação da escrita das informações enviadas ao RTC.
Porta 2 bit7	MGRTCCS#	Pino responsável pela ativação do RTC como o componente ao qual as informações constantes no barramento de dados se destinam.
RXD	MGRX	Utilizado para comunicação síncrona entre o µGerente e o µ Principal. É através deste pino

[3] Representantes da Atmel Corporation no Brasil: HD Sistemas Eletrônicos Ltda e Colgil Ltda.

		que os dados são transmitidos ou recebidos na comunicação entre estes dois microcontroladores.
TXD	MGTX	Pela utilização de um modo de comunicação síncrono entre o µGerente e o µ Principal, neste pino encontra-se presente o sinal de *clock* que fará a sincronização da comunicação entre os dois componentes.
INT 0	INTCLK#	Pino onde chega a informação de que o horário programado no RTC já foi alcançado.
INT 1	MCCLK	Pino onde chega um sinal de interrupção gerado pelo µComunicação para indicar que está presente no pino onde transitam os dados para intercâmbio de informações entre este e o µGerente um novo bit válido.
T0		Não utilizado
T1	MGMPTRANS#	Utilizado como um sinalizador de que o canal serial está ocupado. Por existir uma comunicação entre o µPrincipal e o µGerente pelo sistema *Half-Duplex*, foi necessário a utilização deste sinalizador para evitar que os dois tentem utilizar o canal para transmissão simultaneamente.
WR#	MGCLK	Pino onde sai um sinal de interrupção gerado pelo µGerente para indicar que está presente no pino onde transitam os dados para intercâmbio de informações entre este e o µComunicação um novo bit válido.
RD#	MGMCDAD	Bit por onde circulam os dados entre o µGerente e o µComunicação.

Outra decisão importante do Bloco Administrador foi qual componente seria utilizado como Relógio de Tempo-Real (*Real Time Clock*, RTC) da unidade de processamento e como ele estaria conectado com o resto da unidade.

Como requisitos principais necessários para o RTC, foram impostos:

- A possibilidade de programar um horário e o componente gerar uma interrupção quando o horário for alcançado.
- A granularidade do RTC fosse pelo menos de 1 ms (granularidade desejada para a programação de tarefas do usuário).

Entre os componentes pesquisados, foi escolhido o MM58167 da National como o que melhor atendia os requisitos acima listados. Entre suas características, destacam-se:

- Compatível com microprocessadores com barramento de 8 bits.
- Possibilidade de contagem e comparação de milissegundos até meses.
- 56 bits de RAM para serem armazenados dados destinados à comparação com o contador de tempo real.
- Dois canais de saídas de interrupção com possibilidade de oito diferentes sinais de interrupção.
- Bit de Status de Transição para indicar quando houve uma atualização do valor do relógio durante sua leitura. Esta característica é importante, pois caso houver esta atualização, pode ocorrer em erro de leitura. Por exemplo: Com o relógio marcando 48'59"999, é feita a leitura do milésimo de segundos até os segundos. Quando é realizada a leitura dos minutos, já ocorreu a transição para 49'00"000. Caso não existisse o Bit de Status de Transição, o microcontrolador que tivesse feito a leitura acima interpretaria que o horário atual seria 49'59"999. Com a existência do bit supra referido, basta ao microcontrolador que fez a leitura do horário, efetuar a leitura do bit de Status. Caso ele esteja setado, o microcontrolador dispensa as informações obtidas e repete o procedimento de leitura.

5.3 BLOCO EXECUTOR

O Bloco Executor é formado, fundamentalmente, por um microcontrolador 89C52 que faz o papel do µPrincipal e por uma memória de 128 KBytes. A função dos pinos de *hardware* do microcontrolador é a seguir detalhada:

	NOME RECEBIDO	FUNÇÃO
Porta 0	AD0..AD7	Utilizado como barramento de dados e byte inferior do barramento de endereços destinados ao acesso a memória externa de 128 KBytes e o *Buffer* de Comunicação.
Porta 1	MPP1.0..MPP1.7	Porta livre destinada ao uso pela aplicação do usuário. O acesso a este pino é feito por intermédio do conector CN2 [ver 5.6.2.2]
Porta 2	A8..A15	Utilizado como byte alto do barramento de endereços destinado ao acesso a memória externa de 128 KBytes.
RXD	MPRX	Pela utilização de um modo de comunicação síncrono entre o µGerente e o µ Principal, é através deste pino que os dados são transmitidos ou recebidos na comunicação entre estes dois microcontroladores.
TXD	MPTX	Pela utilização de um modo de comunicação síncrono entre o µGerente e o µ Principal, neste pino encontra-se presente o sinal de *clock* que fará a sincronização da comunicação entre os dois componentes.
INT 0	MPINT0	Entrada de interrupção disponível para ser utilizada pela aplicação do usuário. O acesso a este pino é feito por intermédio do conector CN2 [ver 5.6.2.2].
INT 1	MPINT1	Dependendo da configuração do *jumper* JP2, este

		pino pode ser utilizado como mais uma interrupção disponível para ser utilizada pela aplicação do usuário, ou como receptor do sinal MGMPTRANS# já explicado.
T0	RAMPROG#	Sinal utilizado para possibilitar a escrita na parte da memória de 128 KBytes presente neste bloco destinada a parte de código da aplicação do usuário.
T1	MPFF#	Sinal de entrada que indica que a memória onde são escritas as mensagens destinadas à tarefa já está lotada, portanto não podem ser enviadas mais informações, enquanto o Bloco de Comunicação não ler mais alguns bytes.
WR#	WR#	Sinal que habilita a escrita na parte da memória de 128 KBytes destinada aos dados da aplicação do usuário.
RD#	RD#	Sinal que habilita a leitura na parte da memória de 128 KBytes destinada aos dados da aplicação do usuário.
PSEN#	PSEN#	Sinal que habilita a leitura na parte da memória de 128 KBytes destinada ao código da aplicação do usuário.
ALE	ALE	Sinal que possibilita a multiplexação entre os dados e a parte baixa dos endereços presentes no barramento AD0..AD7

Como memória de 128 KBytes foi utilizada uma RAM não volátil. A escolha por um componente com capacidade de 128 KBytes adveio da decisão de colocar, no mesmo componente, os 64 KBytes de código e os 64 KBytes de dados que o microcontrolador 89C52 tem capacidade de acessar diretamente. A escolha por uma RAM não volátil possibilita que o código da aplicação seja ali armazenado e fique imutável até que se deseje.

A RAM não volátil utilizada foi a DS1245Y da Dallas Semiconductors. Esta se caracteriza por:

- Pelo menos 10 anos de retenção dos dados na falta de energia.
- Proteção automática dos dados durante a queda de energia.
- Pino compatível com as memórias RAM voláteis de 128 KBytes.
- Sem limite de ciclos de escrita.
- Tempo de acesso de 70 ns.

5.4 BLOCO DE COMUNICAÇÃO

O bloco de comunicação é formado por um microcontrolador 89C52 que faz o papel do µComunicação, um circuito eletrônico que implementa a comunicação serial via interface EIA-485, um outro circuito eletrônico que implementa a comunicação serial via interface RS-232 e uma parte lógica que determina qual das duas interfaces seriais disponíveis estará sendo utilizada a cada instante. O chaveamento entre estas duas interfaces é feito por *software*, possibilitando que a troca ocorra dinamicamente.

Abaixo, segue uma descrição de qual função foi atribuída a cada um dos pinos do µComunicação.

	NOME RECEBIDO	FUNÇÃO
Porta 0	MC0..MC7	Utilizado como barramento de dados para envio de informações ao *Buffer* de Comunicação.
Porta 1 bit 0	MCCLK	Pino onde sai um sinal de interrupção, gerado pelo µComunicação, para indicar que está presente no pino onde transitam os dados para intercâmbio de informações entre este e o µGerente, um novo bit válido.
Porta 1 bit 1	MGMCDAD	Bit por onde circulam os dados entre o µGerente e o µComunicação.
Porta 1 bit 2	485EN	Pino cujo sinal habilita a transmissão via interface física serial padrão EIA-485. Esta

		interface caracteriza-se por possibilitar transmissão multiponto, isto é, uma mensagem enviada por este meio físico pode ser recebida e/ou respondida por vários outros equipamentos ligados a este mesmo meio. Uma das exigências do padrão EIA-485 é de que um equipamento conectado ao barramento deve estar em alta impedância sempre que não estiver de posse do barramento. Quando chegar a sua vez de assumir o mesmo, é o pino 485EN que possibilita ao canal de transmissão da UP sair do estado de alta impedância.
Porta 1 bit 3	232RTS	A interface de comunicação serial que segue o padrão RS-232 implementada na UP possibilita que, além dos obrigatórios sinais de RXD e TXD, tenhamos mais dois sinais de controle para que possa ser implementado algum protocolo privativo de comunicação entre a UP e o equipamento a que está estabelecendo comunicação serial. O pino 232RTS é o pino de saída que foi reservado para implementar o protocolo desejado.
Porta 1 bit 4	G485	Pino que determina se a interface serial que está sendo utilizada é a EIA-485 ou RS-232.
Porta 1 bit5 a bit7	MCP1.5 a MCP1.7	Pinos livres destinados ao uso pela aplicação do usuário. O acesso a estes pinos é feito por intermédio do conector CN2. Em conjunto com os pinos da porta 2, pode ser utilizado para acessar um barramento paralelo.
Porta 2	MCP2.0..MCP2.7	Porta livre destinada ao uso pela aplicação do usuário. O acesso a este pino é feito por

		intermédio do conector CN2.
RXD	RXD	Pino responsável pela recepção do sinal oriundo da interface serial ativa no momento.
TXD	TXD	Pino responsável pela transmissão do sinal destinado à interface serial ativa no momento.
INT 0	MCEF	Sinal de interrupção que indica ao µComunicação a existência de pelo menos um byte a ser lido no *Buffer* de Comunicação.
INT 1	MGCLK	Pino onde chega um sinal de interrupção gerado pelo µGerente para indicar que está presente no pino onde transitam os dados para intercâmbio de informações entre este e o µComunicação um novo bit válido.
T0	MCFF#	Sinal de entrada que indica que a memória de recepção de mensagens destinada às tarefas já está lotada, portanto não podem ser enviadas mais informações.
T1	MCHF#	Sinal de entrada que indica que a memória de transmissão de mensagens destinadas às tarefas já está com mais da metade de sua capacidade preenchida. Com esta informação, o µComunicação pode aumentar a prioridade com que efetuará a leitura das mensagens no *Buffer* de Comunicação. Esta ação visa evitar que o mesmo fique lotado, impedindo que o µprincipal possa colocar mais informações no mesmo.
WR#	MCWR#	Sinal que habilita a escrita na memória de recepção de mensagens do *Buffer* de Comunicação.
RD#	MCRD#	Sinal que habilita a leitura da memória de

		transmissão de mensagens do *Buffer* de Comunicação.

Para possibilitar o chaveamento via *software* das duas interfaces físicas de comunicação serial, foi utilizado um multiplexador 2 . X 1 quádruplo, 74HC157, que implementou a lógica mostrada na figura 5-3.

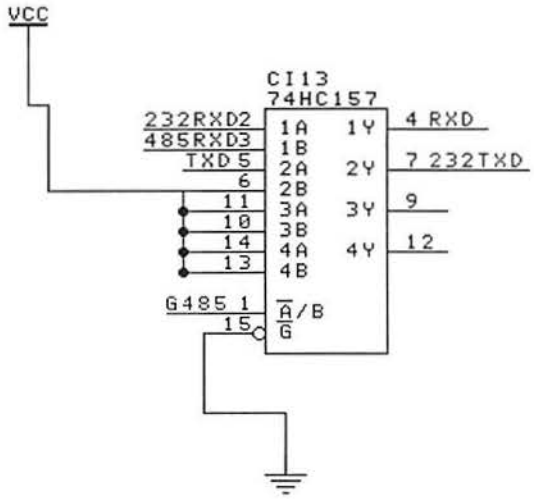


Figura 5-3 *Hardware* para chaveamento das 2 interfaces seriais disponíveis

Quando G485 não estiver ativado, a comunicação dar-se-á pela interface RS232. Para isto, o multiplexador permitirá a passagem do sinal de recepção oriundo desta interface para o RXD do µComunicação e direcionará o TXD deste para o 232TXD. A transmissão pela interface EIA485 não é ativa porque o sinal 485EN, que é o responsável por esta ativação, está ressetado.

Quando G485 está ativado, a comunicação dar-se-á pela interface EIA485. Para isto, o multiplexador permitirá a passagem do sinal de recepção oriundo desta interface para o RXD do µComunicação e fará com que se direcione para o 232TXD um sinal estacionário com nível lógico 1. Com isto fica atendido o protocolo de linha sem transmissão de dados para esta interface. Como já explicado no parágrafo acima, a ativação da transmissão via EIA485 é de responsabilidade do sinal 485EN.

5.5 *BUFFER* DE COMUNICAÇÃO

O *Buffer* de Comunicação possibilita trocas de mensagens entre as tarefas internas a UP, e entre estas e tarefas localizadas em outras Unidades de Processamento.

Sua implementação foi feita através da utilização de duas memórias FIFO Ram fabricadas pela Integrated Devices Technology Inc, DT7202LA. Suas principais características são:

- 1K x 9 bits.
- Utiliza metodologia de liberação dos dados armazenados pela tarefa *First-In/First-Out*, isto é, o primeiro dado armazenado na memória será o primeiro a ser lido.
- Sistema de Dupla-Porta que permite escrita e leitura assíncrona e simultânea.
- Não necessita barramento de endereços para armazenar ou ler os dados, bastando que se ative o sinal de *Write* ou *Read* respectivamente.
- Pinagem e funções compatíveis com a família 720X, possibilitando que se substitua este componente por um produzido por outro fabricante, diminuindo o risco de uma decisão de cancelamento da fabricação deste tipo de componente pela IDT inviabilizar a montagem de novas placas de UPs.
- Completa capacidade de "cascateamento" de componentes, tanto com relação ao número de bits quanto à capacidade de armazenamento de dados.
- Sinalizadores que informam se a memória se encontra lotada, com mais da metade de sua capacidade de armazenamento utilizada ou completamente vazia. O primeiro sinalizador evita a escrita de dados quando não há mais espaço para tanto. O sinalizador de que a memória está completamente vazia evita a leitura de um dado que não está armazenado.

No *hardware* implementado, uma das memórias foi destinada à colocação de mensagens oriundas das tarefas internas a UP que serão despachadas pelo µComunicação ao destino correto, podendo ser uma tarefa interna ou externa a mesma UP. A segunda memória se destina ao recebimento de mensagens que tem por destino tarefas pertencentes à UP em que a memória está inserida. As mensagens podem ser oriundas de tarefas internas ou externas à UP. A Figura 5-4 apresenta estas memórias e os sinais aos quais elas foram interligadas.

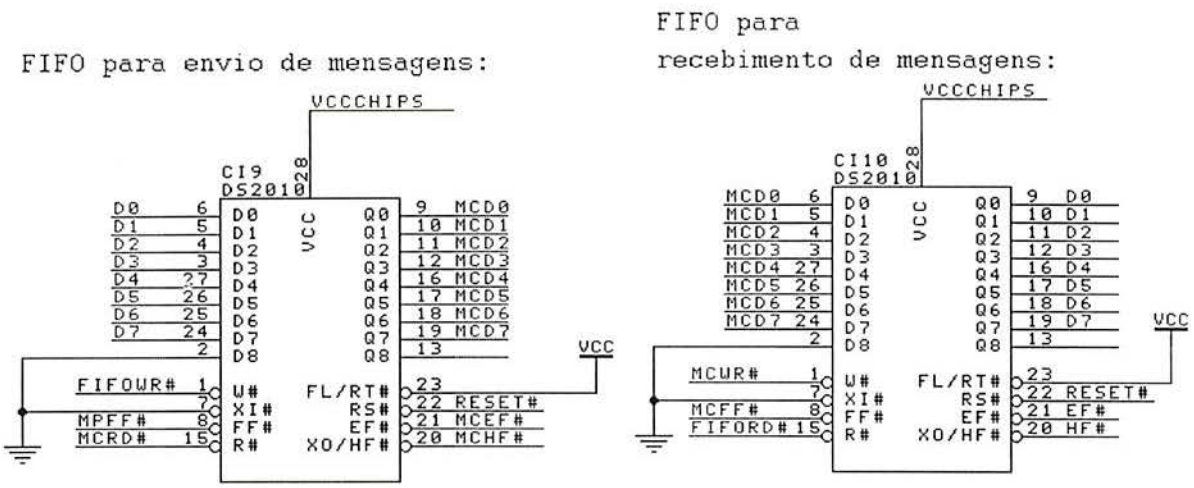


Figura 5-4 FIFOs de envio e recebimento de mensagens

6 PROJETO DE *SOFTWARE*

Sendo o escopo desta dissertação uma arquitetura de *hardware*, para que a mesma pudesse ser validada foi necessário desenvolver e implementar uma placa, com os dispositivos já referidos no capítulo anterior. Só que este *hardware*, por ser microcontrolado, necessita que seja desenvolvido um *software* específico para sua adequada utilização. Por se tratar de uma arquitetura para sistemas tempo-real distribuídos, foi necessário desenvolver o núcleo básico de um sistema operacional que contemple as características de tais tipos de sistema. Como a arquitetura apresentada possui três microcontroladores, foi necessário desenvolver três sistemas operacionais distintos. Cada um responsabilizando-se pelas tarefas destinadas a um respectivo microcontrolador presente na arquitetura projetada. Estes sistemas operacionais implementam as características necessárias para um sistema tempo-real distribuído de modo a possibilitar a utilização da UP por algumas aplicações para tais sistemas.

O fato de ter sido necessário desenvolver três núcleos de sistema operacional ao invés de um somente, como na arquitetura convencional, trouxe uma dificuldade maior para depuração dos três em conjunto. Por outro lado, após os mesmos estarem estáveis, qualquer modificação desejada tende a ser facilitada devido a cada sistema operacional desenvolvido, quando analisado isoladamente, ser mais simples do que o sistema operacional que seria necessário modificar na arquitetura convencional. Além disto, por estarem sendo utilizados três microcontroladores, sobrarão mais recursos destes do que na arquitetura convencional, como mais memória interna, *timers*, portas de entrada/saída de dados, etc. Este fato possibilita maior facilidade para o processamento das tarefas da aplicação.

Salienta-se que os sistemas operacionais desenvolvidos não foram desenvolvidos com o intuito de tornarem-se um produto. Uma vez que o desenvolvimento de um sistema operacional tempo-real distribuído foge ao escopo da presente dissertação, optou-se por algumas simplificações com o objetivo de facilitar a implementação, reduzindo a abrangência de aplicações que possam ser implementadas.

Este capítulo explicará detalhes dos sistemas operacionais desenvolvidos para cada um dos microcontroladores existentes na arquitetura. A nomenclatura das funções que são aqui apresentadas segue a sugestão de padronização de nomenclatura dos *softwares* a serem desenvolvidos no laboratório de Instrumentação Eletro-Eletrônica da UFRGS. Destaca-se a letra minúscula que antecede os nomes das funções. Esta indica o tipo de parâmetro que ela retorna, segundo a seguinte convenção:

- bNomedaFunção: função retorna um bit
- cNomedaFunção: função retorna um parâmetro tipo char
- iNomedaFunção: função retorna um parâmetro tipo inteiro
- NomedaFunção: função não retorna nenhum parâmetro.

A primeira seção deste capítulo tratará da comunicação entre eles. Desta forma, quando estiver sendo explicada a arquitetura de software interna de cada um dos microcontroladores, fica mais fácil entender as rotinas que necessitam utilizar a comunicação entre estes.

6.1 COMUNICAÇÃO ENTRE OS µCONTROLADORES

6.1.1 Comunicação µPrincipal-µGerente:

O µPrincipal e o µGerente se comunicam sincronamente através do canal serial do 89C52, no modo 0, conforme mostrado na Figura 6-1. Este modo de transmissão escolhido visa acelerar a comunicação entre os dois microcontroladores para melhorar o desempenho da UP, pois quando esta está ocorrendo o µPrincipal não está processando as tarefas da aplicação. Ao invés da transmissão por um canal serial, poderia ter sido escolhido uma interface paralela, como por exemplo uma dupla-porta. Esta opção começaria a ser interessante quando o número de informações a serem trocadas entre os dois microcontroladores começasse a crescer. A opção pela comunicação serial, objetivou deixar pelo menos uma porta de entrada/saída de dados livre em cada um dos microcontroladores, e veio acompanhada de uma estrutura de *software* que concentrou grandes trocas de informações entre os microcontroladores na inicialização da UP. Desta forma tem-se que durante a execução das tarefas da aplicação, o tempo perdido fica minimizado.

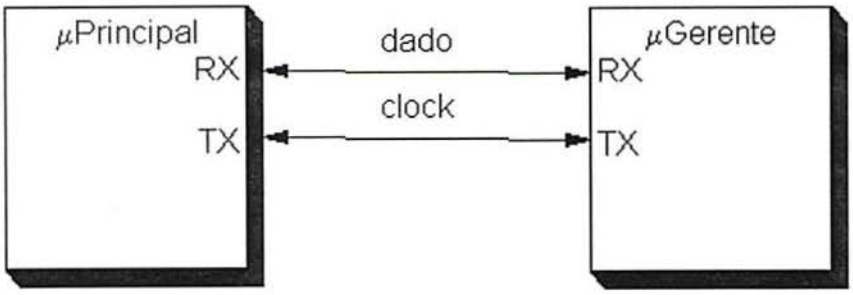


Figura 6-1 Comunicação serial entre os Blocos Executor e Administrador

O µPrincipal pede a transmissão quando deseja:

• Programar um evento: Fornecer ao µGerente as informações necessárias para que este saiba quando cada tarefa estará pronta para execução.

• Cancelar uma tarefa: Retirar a tarefa das filas em que estiver (prontas para execução, esperando eventos síncronos, ...) e zerar o bit correspondente (campo A).

• Avisar o término de execução de uma tarefa: O µGerente envia a próxima tarefa da lista das prontas (retirando-a da lista), reprograma a tarefa que estava executando, caso esta seja cíclica e atualiza a estatística do tempo médio de execução.

• Avisar a suspensão (bloqueio) da execução de uma tarefa: A suspensão de uma tarefa está condicionada à programação de um evento (pode-se programar um evento sem bloquear a tarefa, mas não se pode bloquear uma tarefa sem programar um evento).

A comunicação, no sentido µPrincipal→µGerente, é feita da seguinte maneira:

Os dois bits mais significativos do primeiro byte contém um código dizendo qual das quatro situações citadas acima é a da ocasião. Protocolo:

00 → programação de evento: *PROGR_EV*

01 → término de tarefa: *TERM_TASK*

10 → suspensão de tarefa (com respectiva programação de evento): *SUSP_TASK*

11 → cancelamento de evento/tarefa: *CANC_EV*

Os seis bits menos significativos do primeiro byte são o ID (código identificador) da tarefa correspondente.

No caso da programação de um evento, os próximos bytes são os dados do respectivo evento a ser programado (o tamanho do *frame* não é necessário uma vez que, para cada tipo de evento, a quantidade de dados é uma constante conhecida).

→ No primeiro byte, os três bits mais significativos (7, 6 e 5) informam o tipo de evento a ser programado. O bit 4 diz se o evento será síncrono cíclico (se for 1) ou não. Os bits de 0 a 3 dizem a prioridade base da tarefa (que será inicialmente atribuída também à prioridade dinâmica).

→ Se o evento for um evento síncrono, virão quatro bytes que deverão ser guardados nos campos de horário de ativação. Se o evento for síncrono relativo, existirão mais três bytes que corresponderão ao tempo de ativação relativo ao instante atual.

→ Se o evento for assíncrono, existirá um byte adicional que informará o pino de *hardware* correspondente ao evento (informação contida nos bits menos significativos).

→ Se o evento esperado for uma mensagem de uma outra tarefa, o próximo e último byte informará o ID da tarefa de quem a mensagem deverá vir (um ID igual a zero significa que ele pode receber mensagens de qualquer tarefa).

→ Se o evento for uma ativação relativa, virão mais três bytes que indicarão o pino correspondente ao evento assíncrono (3 bits mais significativos do 1º byte) e o tempo de espera para ativação após o evento assíncrono.

O µGerente toma a iniciativa da transmissão em três situações:

• Quando deseja informar que existem dados disponíveis na FIFO (durante um download), fazendo a intermediação da informação entre o µComunicação e o µPrincipal.

→ Envia um byte com o código 255.

• Quando manda o µPrincipal interromper a execução de uma tarefa qualquer para começar a executar outra mais prioritária, caso o algoritmo de escalonamento seja preemptivo.

→ Envia o byte com o ID da tarefa a ser executada e logo após um byte com o tipo de evento que a fez entrar em execução. Este segundo byte é importante nos casos em que a tarefa poderia ficar pronta para execução por mais de um tipo de evento.

• Quando manda o µPrincipal descarregar mensagens da FIFO (caso particular do caso anterior).

→ Para este tipo de informação apenas um byte de mensagem é enviado. Seu conteúdo já informa as providências que o µPrincipal precisará tomar, quais sejam:

→ Byte 254: retira uma mensagem da FIFO

→ Byte 253: retira duas mensagens da FIFO

→ Byte 252: retira três mensagens da FIFO

Um cuidado especial deve ser tomado no que diz respeito à utilização do canal serial, uma vez que no modo 0 os dados são transmitidos e recebidos pelo mesmo pino. Assim, uma situação onde ambos os µcontroladores desejem transmitir ao mesmo tempo pode ser desastrosa. Há um pino especialmente destinado a sincronização da comunicação (MGMPTRANS) que é ressetado sempre que uma comunicação é iniciada e setado novamente quando esta acaba. Porém, ainda existe a situação dos dois checarem o pino ao mesmo tempo, verem que o canal está livre e iniciarem uma transmissão simultânea.

Para tanto, foi adotado o seguinte protocolo: o µPrincipal pode tomar a iniciativa da comunicação quando desejar (tem a prioridade) e sempre espera uma resposta do µGerente (ACK). O µGerente, sempre que quiser transmitir (tomando a iniciativa) deve esperar tempo suficiente para que, se uma comunicação vinda do µPrincipal foi iniciada, este tenha tempo de enviar pelo menos um byte ao µGerente, possibilitando a este reconhecer que a linha não está mais disponível para transmissão. A prioridade dada ao µPrincipal advém do fato de que quando este resolve se comunicar, a resposta deve ser a mais imediata possível, uma vez que nas comunicações iniciadas por ele inclui-se a que informa que a tarefa que ele estava processando já terminou ou está bloqueada. Portanto, o µPrincipal está momentaneamente sem atividade, fato que, por filosofia da arquitetura, deve ser abreviado ao máximo.

6.1.2 Comunicação µComunicação-µGerente:

O µComunicação e o µGerente se comunicam através de pinos especialmente reservados para este fim, de forma a gerarem uma interrupção a cada bit transmitido (comunicação lenta).

O µComunicação pede a transmissão quando deseja:

1. Informar a chegada de uma mensagem para alguma tarefa.
2. Sincronizar o relógio.

3. Informar o download de um novo aplicativo

O µGerente inicia a comunicação apenas quando deseja responder ao µComunicação sobre o horário atual do RTC (sincronização).

Eles utilizam um protocolo próprio, através de pinos reservados para este fim. Quem inicia a comunicação (transmissor) deve dar um pulso e esperar dois pulsos de resposta do receptor para considerar a comunicação iniciada.

A espera dos dois pulsos de resposta é temporizada pelo timer 1, de forma a não acontecer de ambos os µcontroladores tentarem iniciar uma comunicação ao mesmo tempo e ficarem esperando para sempre a resposta um do outro (situação de *deadlock*). Nesse caso, o µGerente tem a preferência da retransmissão, podendo, a partir de então, transmitir seus dados para o receptor com um pulso em MCCLK, que a cada bit recebido responderá com uma transição no pino MGCLK. Após o último bit ser transmitido, o receptor deve devolver novamente dois pulsos indicando comunicação finalizada OK.

Deve-se salientar que sempre que um dos lados terminar um processo de transmissão, este deve colocar o pino MGMCDAD em nível lógico "1", possibilitando assim que o bit da porta possa ser utilizado também como entrada de dados.

6.1.3 Comunicação µPrincipal-µComunicação:

Esta é feita por intermédio da FIFO Ram e constitui-se basicamente de mensagens entre as tarefas. O µPrincipal escreve os dados na FIFO, gerando uma interrupção no µComunicação. O µPrincipal, entretanto, apenas lê dados da FIFO quando ordenado pelo µGerente.

6.2 µGERENTE

6.2.1 Visão Geral

O sistema operacional implementado no µGerente, baseia-se no gerenciamento de quatro tabelas de dados que são constantemente atualizadas e consultadas pelo sistema operacional, para que este tome as decisões e providências necessárias para que processamento da aplicação na Unidade de Processamento siga sem problemas.

As principais funções do sistema operacional do µGerente são:

- Escalonamento das tarefas prontas para execução
- Detecção de eventos assíncronos
- Chegada de um determinado tempo que modifica o estado de uma tarefa
- Chegada de uma mensagem
- Bloqueio de tarefas até que algum evento ocorra

Para que estes e outros serviços fossem possíveis, algumas tabelas e serviços foram implementados. A Figura 6-2 mostra a relação entre estes.

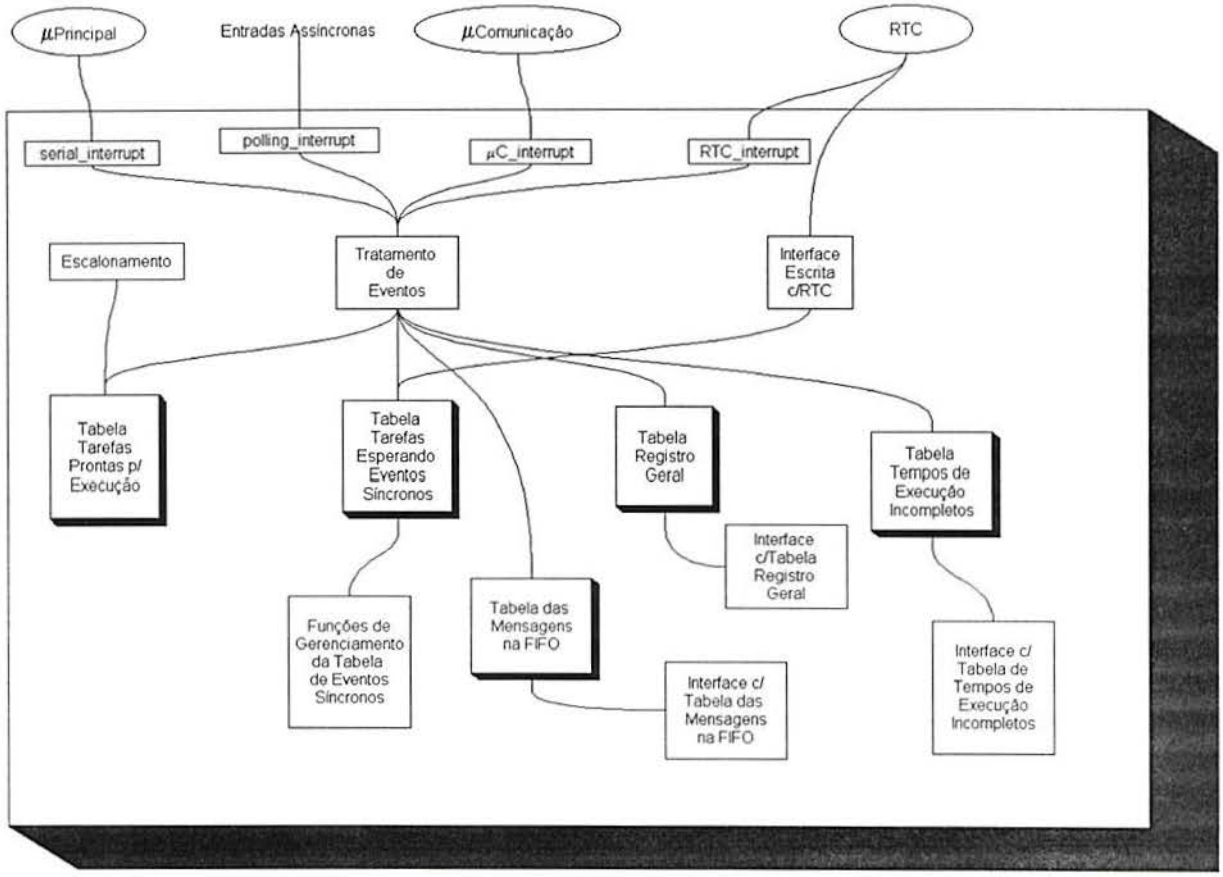


Figura 6-2 Estrutura interna do Bloco Administrador

O fluxograma da Figura 6-3 mostra como o núcleo do Bloco Administrador foi implementado.

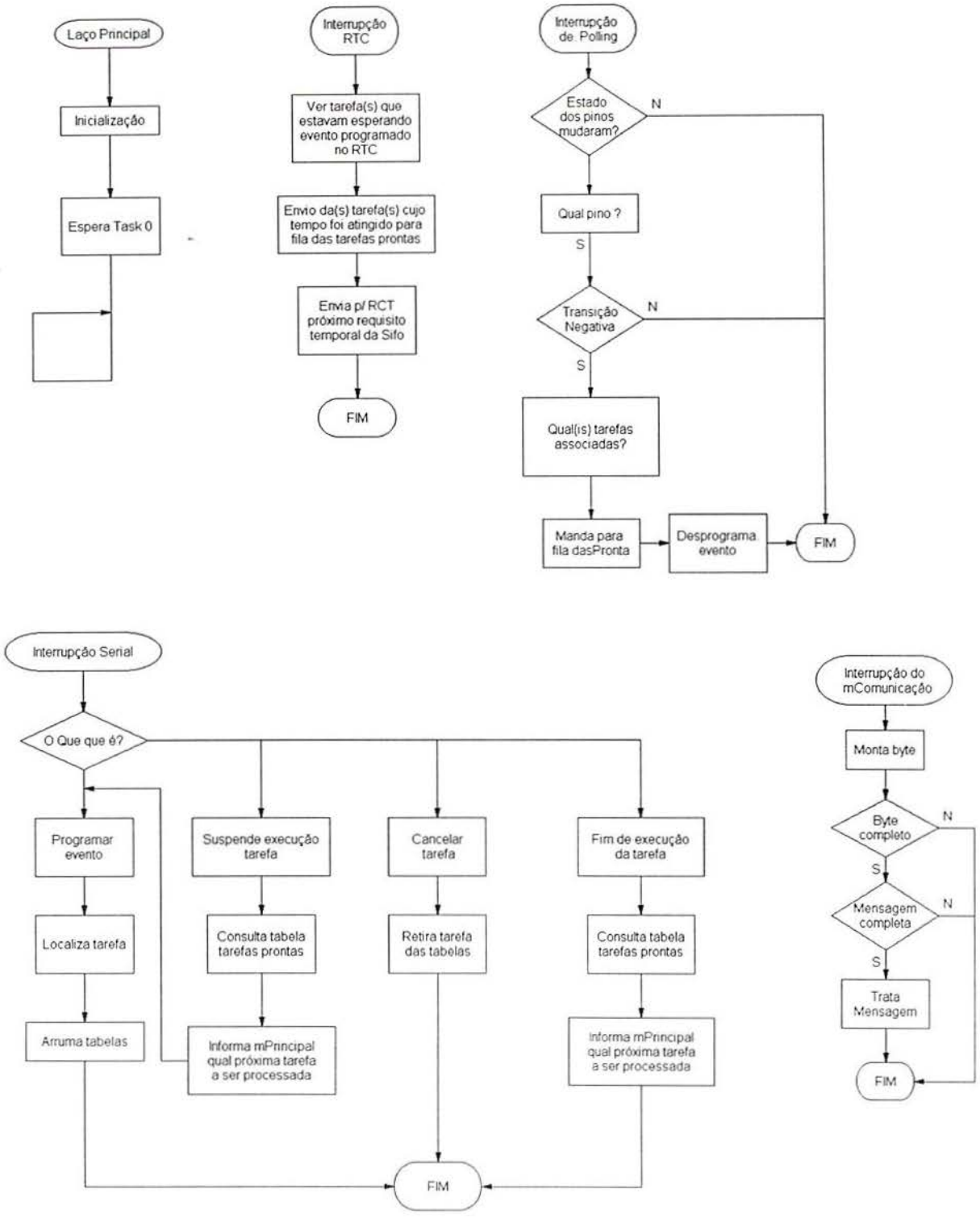


Figura 6-3 Fluxograma do Bloco Admininstrador

6.2.2 Inicialização

Quando da inicialização da Unidade de Processamento, o µGerente recebe do µPrincipal as seguintes informações:

• Código Identificador de todas as tarefas existentes na UP

• Dependência das tarefas, isto é, quais e de que tipo (assíncronos, síncronos absolutos, síncronos relativos ou mensagens) são os eventos que cada uma das tarefas irá esperar para se tornarem ativas.

• *Deadline* das tarefas, isto é, qual o limite máximo de tempo que uma tarefa poderá esperar para ter completado o processamento de sua instância, após a instância ter sido considerada pronta para execução.

6.2.3 Recursos:

O µGerente utiliza os seguintes recursos de sua estrutura interna:

timer 0 → temporizar o *polling* dos eventos assíncronos (timer no modo 2).

timer 1 → temporizar o *time-out* do µComunicação na comunicação entre ambos (timer no modo 1).

interrupção externa 0 → ativada pelo RTC no caso de um evento síncrono.

interrupção externa 1 → gerada pelo µComunicação na transmissão de um bit para o µGerente.

canal serial → opera no modo 0 e faz a comunicação com o µPrincipal.

6.2.4 Serviços do Sistema Operacional:

Os serviços implementados no sistema operacional do µGerente, podem ser divididos nos seguintes grupos:

6.2.4.1 Serviços de Interface de escrita com o RTC

6.2.4.1.1 Programação do RTC

Serviço que ao receber um horário de ativação, programa o RTC para gerar uma interrupção quando este instante chegar. Implementado pela função "ChangeRTC".

6.2.4.2 Serviços de Escalonamento

6.2.4.2.1 Algoritmo de escalonamento

Serviço que implementa o algoritmo de escalonamento responsável pela decisão de qual tarefa estará sendo processada no µPrincipal. A estrutura do sistema operacional permite a troca deste algoritmo, facilitando testes e pesquisas sobre a eficácia de diferentes algoritmos de escalonamentos. É chamada pela função Insert Ready Task que envia duas tarefas para que o algoritmo de escalonamento decida qual deve ter acesso ao µPrincipal primeiramente. Implementado pela função "Scheduling Algorithms".

6.2.4.2.2 Inserir novas tarefas prontas para execução na fila apropriada

Responsável por inserir novas tarefas na fila de tarefas prontas para execução seguindo resultado de consulta ao algoritmo de escalonamento. Para decidir onde colocar a mesma, toma a tarefa mais prioritária já presente nesta fila e chama a função Scheduling Algorithms, passando-lhe esta tarefa e a nova tarefa pronta para execução, esperando como resposta qual a mais prioritária neste instante. Caso a nova tarefa pronta não seja a mais prioritária, repete-se o procedimento entre esta e a segunda tarefa da fila das tarefas prontas, e assim sucessivamente até encontrar o lugar correto nesta fila para a nova tarefa que ficou pronta para execução. Implementado pela função "InsertReadyTask".

6.2.4.3 Serviço de Interface com a Tabela-Registro Geral

6.2.4.3.1 Achar localização de uma dada tarefa na tabela-registro de tarefas.

É utilizada por todas as funções do sistema operacional do µGerente que necessitam de informações e/ou modificar algum dado das tarefas da aplicação. Implementado pela função "cWhereIsTask".

6.2.4.3.2 Cancelar espera por eventos assíncronos

Procura no registro geral todas as instâncias de uma dada tarefa e cancela a programação de todos os eventos assíncronos. Não exclui a programação de um evento síncrono que porventura a tarefa esteja esperando. Eventos síncronos são excluídos após o retorno desta função, pela função que a chamar (por exemplo: a chegada de algum evento). A função original sabe que tem que realizar esta tarefa pela consulta obrigatória a um *flag*

específico. Esta atitude foi tomada de modo a facilitar a implementação do sistema operacional. Implementado pela função "bWaitingEventReset".

6.2.4.4 Serviços de Interface com a Tabela de Tempos de Execução Incompletos

6.2.4.4.1 Aquisição do tempo parcial de execução de uma tarefa

Busca, na tabela de tempos de execução, o tempo já executado por esta instância da tarefa solicitada, retornando este dado. Pode ser utilizado por algoritmos de escalonamento mais elaborados que consideram o tempo de execução médio das tarefas e o tempo que uma tarefa já está executando para decidir sobre realizar ou não uma preempção na mesma. Implementado pela função "iGetCurrentExecTime".

6.2.4.4.2 Colocação do tempo de execução da tarefa.

Insere o tempo atual de execução (fornecido) da tarefa dada na tabela de tempos de execução. Implementado pela função "PutExecTime".

6.2.4.5 Serviços de Tratamento de Eventos

6.2.4.5.1 Chegada de mensagens

Serviço chamado quando o µGerente, através do µComunicação, fica sabendo que há na FIFO uma mensagem endereçada a uma tarefa local. A rotina procura, na tabela registro geral, se esta tarefa está esperando alguma mensagem e se o remetente da mensagem esperada é o mesmo da que chegou. Se for, esta tarefa é colocada na lista de tarefas prontas para execução e, se porventura estiver esperando algum outro evento, este evento é desprogramado. Se a tarefa não estiver esperando mensagem alguma do remetente em questão, a informação da existência da mensagem é armazenada na tabela de mensagens na FIFO RAM. Implementado pela função "MessageArrived".

6.2.4.5.2 Atendimento as informações oriundas do µPrincipal

Serviço que analisa os dados recebidos do µPrincipal, verificando se houve a programação de um evento, o término da tarefa que estava executando ou o cancelamento de alguma tarefa. Para cada caso, são atualizadas nesta rotina todas as tabelas pertinentes. Implementado pela função "SerialIntDealing".

6.2.4.5.3 Programação do RTC

Serviço chamado após o acontecimento de um evento síncrono informado pelo RTC. Envia a primeira tarefa da fila de tarefas que esperam eventos síncronos para a lista de tarefas prontas para execução. Se for uma tarefa cíclica, esta deve ser reprogramada para a espera de outro evento síncrono com o tempo relativo da mesma em relação ao instante atual. As próximas tarefas terão seus instantes de próxima ativação checados e, caso o horário de próxima ativação seja muito pequeno em relação ao atual (menor que BONUS_TIME), elas não são programadas no RTC mas são enviadas diretamente à lista de tarefas prontas para execução. Implementado pela função "RTCIntDealing".

6.2.4.5.4 Controle da chegada de eventos assíncronos

Serviço chamado após ser feita a leitura da porta correspondente aos eventos assíncronos (*polling*), tem por objetivo verificar se houve nos pinos desta uma transição de nível 1 para nível 0, caracterizando um evento. Em caso positivo, procura na tabela apropriada quais tarefas estavam esperando por este evento e as coloca na lista de tarefas prontas para execução. Implementado pela função "AsyncEventDealing".

6.2.4.6 Serviços de Gerenciamento da Tabela de Eventos Síncronos

6.2.4.6.1 Conversão dos valores de tempo

Transforma dados com informação de tempo (de minutos à milésimos de segundo) lidas diretamente do RTC para um formato binário, facilmente manipulável pelo µcontrolador. Implementado pela função "iBCD_to_Binary".

6.2.4.6.2 Comparação de horários de ativação

Compara duas estruturas com horário de ativação e verifica qual das duas acontece antes. Implementado pela função "bCompare".

6.2.4.6.3 Organização da fila de eventos síncronos

Insere/reordena a fila de eventos síncronos, mantendo-a ordenada conforme o momento de ativação das tarefas. Implementado pela função "RefreshSyncTable".

6.2.4.6.4 Adequação dos valores dos horários de ativação

Usada para gerar os horários de ativação (absolutos) a partir dos tempos relativos de próxima ativação fornecidos (ativar daqui a x segundos...). Serve tanto para tarefas que dependam de eventos síncronos relativos quanto para tarefas que dependam de ativação temporal relativa a chegada de um evento assíncrono. Implementado pela função "PutAbsoluteTime".

6.2.4.7 Serviço de Interface com Tabela das Mensagens na FIFO

6.2.4.7.1 Procura de mensagens existentes

Procura na tabela de mensagens na FIFO, se há uma mensagem armazenada para uma certa tarefa cujo remetente seja uma outra tarefa específica. Se houver, devolve a posição desta mensagem na FIFO. Utilizada quando o µPrincipal informar que uma tarefa foi bloqueada a espera de uma mensagem de uma tarefa específica. Implementado pela função "cSearchMsgInFIFO".

6.2.4.8 Serviços de Atendimento a Interrupção

6.2.4.8.1 Interrupção Serial

Rotina da interrupção serial, responsável pela comunicação com o µPrincipal. A interrupção é gerada apenas quando o µGerente recebe o primeiro bit a ser enviado pelo µPrincipal (ver seção *Comunicação entre os µControladores*); o complemento da transmissão de dados é feita sem a geração de interrupção. Implementado pela função "serial_interrupt".

6.2.4.8.2 Interrupção do RTC

Rotina da interrupção externa 0 (pino 12), provocada pelo RTC para avisar que chegou o horário previamente programado, gerando a ativação da tarefa que estava esperando a chegada deste horário. Seta o *flag ExtInterrupt0*. Implementado pela função "RTC_interrupt".

6.2.4.8.3 Interrupção do µComunicação

Rotina da interrupção externa 1 (pino 13). É causada pelo µComunicação para avisar que existe um bit disponível no pino 17 (MGMCDAD) referente à transmissão de

dados deste para o µGerente ou para avisar que recebeu o bit enviado anteriormente pelo µGerente (ver Seção *Comunicação entre os µControladores*). Implementado pela função "µC_interrupt".

6.2.4.8.4 Interrupção para verificação do *polling*

Rotina da interrupção do timer 0, responsável pelo *polling* da porta P1 à procura de transições nos pinos (eventos assíncronos). Se houver uma transição em ao menos um pino, seta o *flag ExtAsyncEvent*, para que esta transição seja melhor verificada posteriormente na função *AsyncEventDealing*. Esta verificação não é feita nesta mesma função para evitar que o microcontrolador fique muito tempo processando um atendimento a interrupção. Implementado pela função "polling_interrupt".

6.2.4.8.5 Interrupção sobre tempo de espera para *handshaking* entre µGerente-µComunicação

Rotina de interrupção do timer 1, responsável pela temporização do tempo de espera entre os dois pulsos de resposta definidos para o início da comunicação entre µGerente-µComunicação. Implementado pela função "timer_interrupt".

6.2.4.9 Serviço de Inicialização de uma Nova Aplicação do Usuário

6.2.4.9.1 Reinicialização do µGerente

Rotina chamada ao final de um *download* de uma nova aplicação do usuário. Serve para reinicializar adequadamente o µGerente. Os sistemas operacionais foram desenvolvidos para que possibilitassem este tipo de operação via canal serial RS-232. Isto facilita a implementação de modificações nas aplicações que rodarão na UP, pois não é necessário desconectá-la do sistema, nem retirar qualquer componente da mesma. Implementado pela função "Restart".

6.2.5 Tabelas

No µGerente existem diversas tabelas de dados para a correta organização da gerência das tarefas. São elas:

• **Tabela-registro permanente de tarefas**: Contém todas as tarefas e seus respectivos dados, independente do estado atual. Tabela de formação estática, isto é, as tarefas que a compõe são inicializadas junto com a inicialização da UP. A cada tarefa estão relacionados de 7 a 14 bytes, dependendo da natureza do evento a que esta está vinculada, podendo este evento ser assíncrono, mensagem, síncrono absoluto, síncrono relativo ou uma ativação relativa, como relatado no item 6.1.2.1. Destaca-se que todos os eventos que tenham requisito síncrono, este pode variar de dias a milissegundos, dependendo da necessidade da aplicação.

• **Tabela de tarefas prontas para execução**: Contém o endereço das tarefas na tabela-registro que estão prontas para executar. A ordenação das tarefas nesta tabela dependerá dos critérios adotados pelo algoritmo de escalonamento que está sendo utilizado. Possui uma característica dinâmica, isto é, as tarefas que nela estão presentes dependerão de condições instantâneas, sendo acrescentadas ou retiradas tarefas desta tabela com o acontecimento de um novo evento ou com o término ou bloqueio de tarefa pelo µPrincipal. A cada tarefa é atribuído 1 byte de informação.

• **Tabela de tarefas que esperam eventos síncronos**: Contém ponteiros para os campos de horário de ativação absoluto das tarefas que estão esperando eventos síncronos. Possui característica dinâmica, sendo atribuído 1 byte por tarefa.

• **Tabela de tempos de execução "incompletos"**: Armazena o tempo de execução da tarefa até o momento em que foi bloqueada. Tabela com característica dinâmica, sendo 3 bytes alocados para cada tarefa que tem seu tempo de execução registrado.

• **Tabela das mensagens na FIFO Ram**: Contém o estado atual da FIFO, com registro da tarefa destino e tarefa origem de cada mensagem armazenada. Possui característica dinâmica, sendo que 2 bytes são utilizados para registrar as informações de cada mensagem existente na FIFO Ram.

Todas as tarefas necessariamente constam na primeira tabela (por isso chamada de registro), mas apenas aparecem nas demais tabelas as tarefas que no momento necessitam ser referenciadas.

6.2.5.1 Tabela Registro Geral:

Na tabela-registro, cada tarefa é representada pelos seguintes campos de bits:

Campo	Núm de bits	O que representa
A	1	Está esperando algum evento assíncrono ou mensagem? (1=sim, 0=não)
B	1	Tarefa é cíclica? (1=sim, 0=não)
C	6	Código de identificação da tarefa.
D	4	Prioridade base. (0000=menor prior. 1111=maior prior.)
E	4	Prioridade dinâmica.
F	3	Dependência da tarefa. (ver detalhe ainda neste item)
G	16	Tempo médio de execução (no máximo 1 min).
H	1	Tempo de execução atual está armazenado na tabela de tempos de execução incompletos? (1=sim, 0=não)
I	1	Há alguma mensagem armazenada na FIFO? (1=sim, 0=não)
J	16	*Deadline* da tarefa (no máximo 1min).

Os campos C, F e J são estáticos e são definidos durante o download do aplicativo. O restante pode ser modificado durante o decorrer do tempo pelo sistema operacional (campos A, E, G, H, I) ou pelo aplicativo, ao ser programado um novo evento (campos B, D e outros campos a serem discutidos mais adiante).

O campo A não se refere a tarefas síncronas nem a atuações relativas que esperem eventos síncronos. Isto deve-se ao fato de ter sido considerado mais fácil para gerenciamento do sistema operacional, isolar em uma tabela única e específica toda informação que tiver relação com o RTC. pois todos estes casos estão contidos em uma lista privativa), mas somente às outras tarefas. Já o campo B diz respeito apenas a tarefas síncronas, que podem ser cíclicas ou não. Tarefas que esperam por outros tipos de evento nunca têm este campo setado.

A prioridade base da tarefa, campo D, é a prioridade da instância que foi programada para esta tarefa. É definida quando a aplicação do usuário relaciona a ativação de uma tarefa a chegada de um evento específico. A prioridade dinâmica, campo E, inicialmente é idêntica a prioridade base. Com o passar do tempo, pode ser modificada pelo algoritmo de escalonamento, segundo critérios próprios.

O campo F (dependência da tarefa) pode conter os seguintes códigos:

000 → Não depende de evento algum (é ativada no início do programa ou por alguma outra tarefa). Nesses casos, o campo A da tarefa é sempre 0.

001 → Depende de evento síncrono.

010 → Depende de evento assíncrono externo.

011 → Depende de mensagem de outra tarefa.

100 → Ativação relativa: depende de evento assíncrono externo.

101 → Ativação relativa: depende de evento síncrono.

Define-se por ativação relativa a espera de um evento síncrono condicionada à chegada de um evento assíncrono. O tempo máximo de espera de uma tarefa com ativação relativa está limitado em 32 minutos. Esta dependência se refere a uma mesma tarefa em momentos diferentes, dependendo apenas do tipo de evento que está esperando no momento.

O campo J indica qual o tempo máximo que a tarefa pode levar para terminar sua execução, após ter ingressado na fila das tarefas prontas para execução. Os algoritmos de escalonamento "*Earliest Deadline First*" e "*Least Slack*" necessitam deste campo para serem implementados.

Existem ainda outros campos de bits que são anexados dependendo do tipo de dependência da tarefa em questão. São eles:

	Campo	**Núm bits**	**O que representa**
Eventos que dependem do tempo	L	32	Tempo absoluto da próxima ativação (até 1 mês)
Evento assíncrono	M	3	Pino do *hardware* correspondente (0 a 7)
Mensagem	N	6	Remetente (ID da tarefa-origem)
Evento síncrono cíclico	O	27	Tempo relativo entre as ativações (até 1 dia)
Ativação Relativa	P	3	Pino do *hardware* correspondente (0 a 7)
Ativação Relativa	Q	21	Tempo de espera entre a ocorrência do evento assíncrono e a ativação do evento (< 32 min)

No caso de um evento síncrono cíclico, o campo O ocupa ainda três bits do segundo byte deste campo da mensagem, onde se situa a dependência da tarefa. Se for outro tipo de evento, este campo é deixado vazio.

OBS.: Só podem ser definidas como cíclicas tarefas que esperem eventos síncronos.

Dessa forma, a seguinte representação nos informa como cada tarefa é armazenada nesta tabela. Cada letra indica o campo cuja informação armazenada está relacionada.

byte 1: ABCCCCCC

byte 2: $FFFHIO_2O_2O_2$

byte 3: DDDDEEEE

byte 4: $G_1G_1G_1G_1G_1G_1G_2G_2$

byte 5: $G_2G_2G_2G_2G_2G_2G_2G_2$

byte 6: $J_1J_1J_1J_1J_1J_1J_2J_2$

byte 7: $J_2J_2J_2J_2J_2J_2J_2J_2$

OBS.: Os campos G e J estão subdivididos em 1 e 2, onde os campos 1 contém os segundos e os campos 2 os milissegundos. A inclusão do campo O junto ao byte 2 é relatada adiante.

Se a tarefa depende de algum evento para ser ativada, seguem a estes 7 bytes mais alguns que contém dados relativos a esta ativação. Nesta representação a letra X indica bit não utilizado. São estes:

evento assíncrono: XXXXXMMM

mensagem: XXNNNNNN

evento síncrono absoluto: $L_1L_1L_1L_1L_1L_2L_2L_2$

$L_2L_2L_3L_3L_3L_3L_3L_3$

$L_4L_4L_4L_4L_4L_4L_5L_5$

$L_5L_5L_5L_5L_5L_5L_5L_5$

evento síncrono relativo (cíclico): $L_1L_1L_1L_1L_1L_2L_2L_2$

$L_2L_2L_3L_3L_3L_3L_3L_3$

$L_4L_4L_4L_4L_4L_4L_5L_5$

$L_5L_5L_5L_5L_5L_5L_5L_5$

$O_2O_2O_3O_3O_3O_3O_3O_3$

$O_4O_4O_4O_4O_4O_4O_5O_5$

$O_5O_5O_5O_5O_5O_5O_5O_5$

ativação relativa: $L_1L_1L_1L_1L_1L_2L_2L_2$

$L_2L_2L_3L_3L_3L_3L_3L_3$

$L_4L_4L_4L_4L_4L_4L_5L_5$

$L_5L_5L_5L_5L_5L_5L_5L_5$

$PPPQ_3Q_3Q_3Q_3Q_3$

$Q_4Q_4Q_4Q_4Q_4Q_4Q_5Q_5$

$Q_5Q_5Q_5Q_5Q_5Q_5Q_5Q_5$

Na representação acima apresentada, os subcampos "1" contém informação de dias, os "2" de horas, os "3" de minutos, os "4" de segundos e os "5" de milissegundos. Estes subcampos são enviados compactados em quatro bytes para que a transmissão serial necessária seja mais curta.

Esta tabela contém um byte com "FFh" depois do último elemento. Este indica o fim das informações da tabela de registro.

6.2.5.2 Tabela de tempos de execução incompletos:

Cada vez que uma tarefa é bloqueada por um motivo qualquer, o tempo de execução computado até o momento do bloqueio é armazenado nesta tabela e o bit respectivo (campo H no registro geral) é setado. Quando a tarefa volta à execução, este tempo continua a ser incrementado. Não há ordenamento nesta tabela. Para cada tarefa bloqueada, temos as seguintes informações armazenadas:

ID da tarefa (6 bits)

segundos (6 bits)

milésimos de segundo (10 bits)

Assim, são armazenadas tarefas onde o tempo máximo de execução destas é um minuto. Além disso, há uma variável com o tamanho atual da tabela.

6.2.5.3 Tabela das mensagens na FIFO:

Cada vez que o µComunicação avisa o µGerente da chegada de uma mensagem para uma tarefa qualquer, este deve atualizar esta tabela inserindo a tarefa destino e a tarefa origem na mesma, além de setar o bit correspondente no registro geral (campo I). O µGerente deve ter o controle das mensagens destinadas às tarefas para que possa desbloquear aquelas que esperam mensagens de outras para continuarem a serem processadas.

Portanto, para cada mensagem na FIFO, existem dois bytes: um com a tarefa destino e outro com a tarefa origem. O número de mensagens armazenadas na tabela (e, portanto, na FIFO) é armazenado em uma variável.

6.2.5.4 Tabela das tarefas prontas para execução:

Esta tabela contém as tarefas que estão esperando para serem executadas. A tarefa atualmente em execução não está nesta tabela, mas em um outro registro (GpcExecutingTask → ponteiro para o 1º byte da mesma). Existe uma variável com o número atual de tarefas nesta tabela.

6.2.6 Eventos Síncronos

Os eventos dependentes de horário têm a sua ativação programada no RTC (real-time clock). Como o RTC só permite que um evento seja programado de cada vez, este deverá sempre ser aquele cuja ativação seja a mais próxima do momento atual.

Para a organização da ordem da programação de eventos, temos uma tabela (fila) com ponteiros para as tarefas que esperam por eventos síncronos para se tornarem prontas para execução. Esta tabela está ordenada por horário de ativação.

Dessa forma, eis o que acontece no caso de:

• *Programação de um novo evento síncrono com requisitos temporais absolutos:* É varrida a fila de eventos síncronos programados, que estão ordenados por instante de ativação, e verifica-se o lugar que este novo evento deve ocupar. Caso ele não seja o primeiro da fila, este é inserido no lugar correto e a fila é reordenada. Caso ele seja o primeiro, verifica-se o horário atual. Caso o evento já tenha ocorrido, ele é posto como último elemento da fila.

Caso ainda não tenha ocorrido, verifica-se ainda se o horário de ativação não está muito próximo do horário atual (não compensa programá-lo no RTC), caso em que este é posto diretamente na lista de tarefas prontas. No caso de ter o horário de ativação igual ao de alguma outra tarefa já programada, a nova tarefa é posta na fila após a de mesmo horário. Neste caso, quando o requisito temporal das tarefas é alcançado, ambas são transferidas conjuntamente para a lista de tarefas prontas para execução.

• *Programação de um novo evento síncrono com requisitos temporais relativos:* É solicitado o horário atual para o RTC. Com este, o requisito temporal relativo é transformado em absoluto. A seguir ele é tratado como tal, seguindo a lógica abordada no parágrafo anterior.

• *Chegada de um evento:* Neste caso, manda-se para a lista das prontas a(s) tarefa(s) correspondentes ao evento e, no caso de serem cíclicas, deve-se reprogramá-las. Logo após, reordena-se a fila e programa-se o RTC para o novo horário de ativação.

• *Cancelamento de um evento:* Deve-se retirar a tarefa em questão de todas as listas em que estiver, menos do registro geral. Se o campo/bit de espera de evento estiver setado, deve-se zerá-lo.

6.3 µPRINCIPAL

6.3.1 Visão Geral

Para que o objetivo que originou o desenvolvimento da arquitetura proposta nesta dissertação fosse alcançado, o sistema operacional deste µControlador foi desenvolvido de modo a permitir que fosse minimizado o tempo gasto com tarefas que não o processamento das tarefas da aplicação. Suas funções destinam-se a:

• Realizar o chaveamento de contexto entre a tarefa que estava executando e a nova tarefa determinada pelo µGerente.

• Indicação ao µGerente de quais eventos estão relacionados a cada uma das tarefas que estão bloqueadas. Os eventos podem ser síncronos, assíncronos, uma mensagem ou o simples desbloqueio de uma tarefa específica.

• Transposição de mensagens da FIFO de recebimento de mensagens para uma área específica da memória RAM de dados externa ao µPrincipal. Esta tarefa só é executada quando ordenada pelo µGerente.

A Figura 6-4 demonstra a relação entre as tabelas e serviços implementados no Bloco Executor.

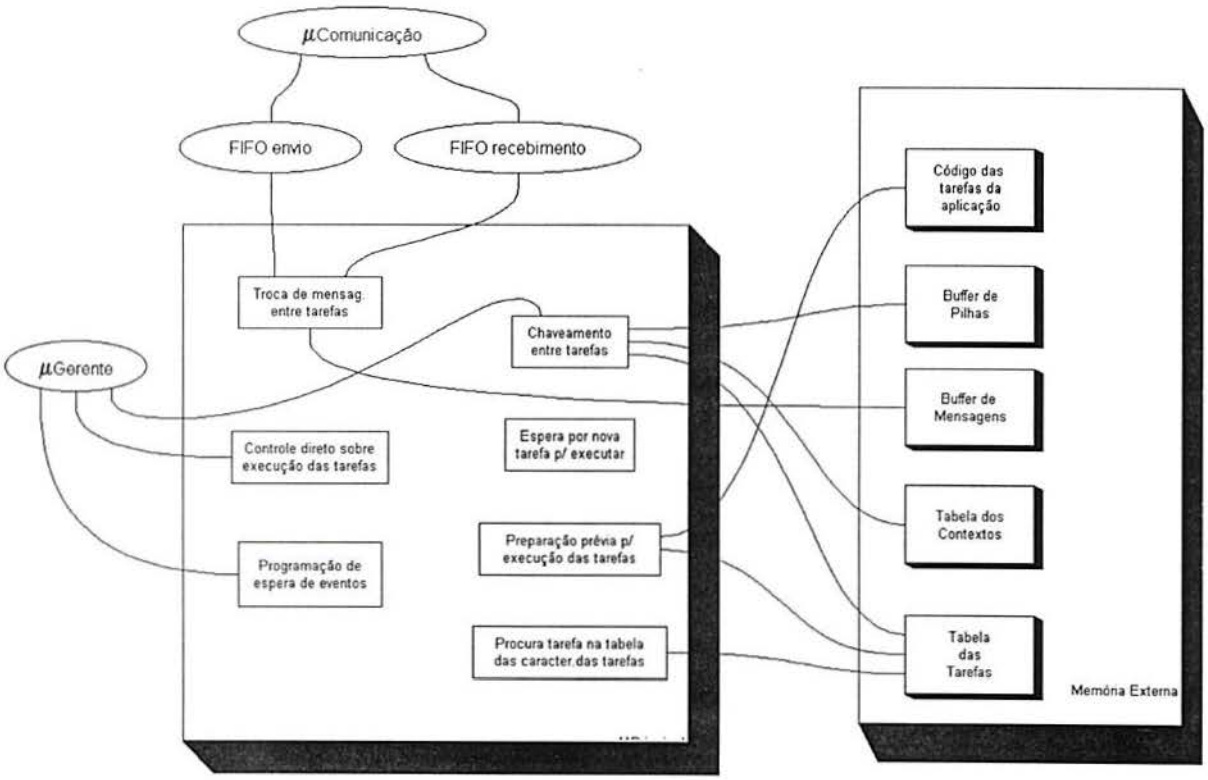


Figura 6-4 Estrutura Interna do Bloco Executor

O fluxograma da Figura 6-5 mostra como o núcleo do Bloco Executor foi implementado.

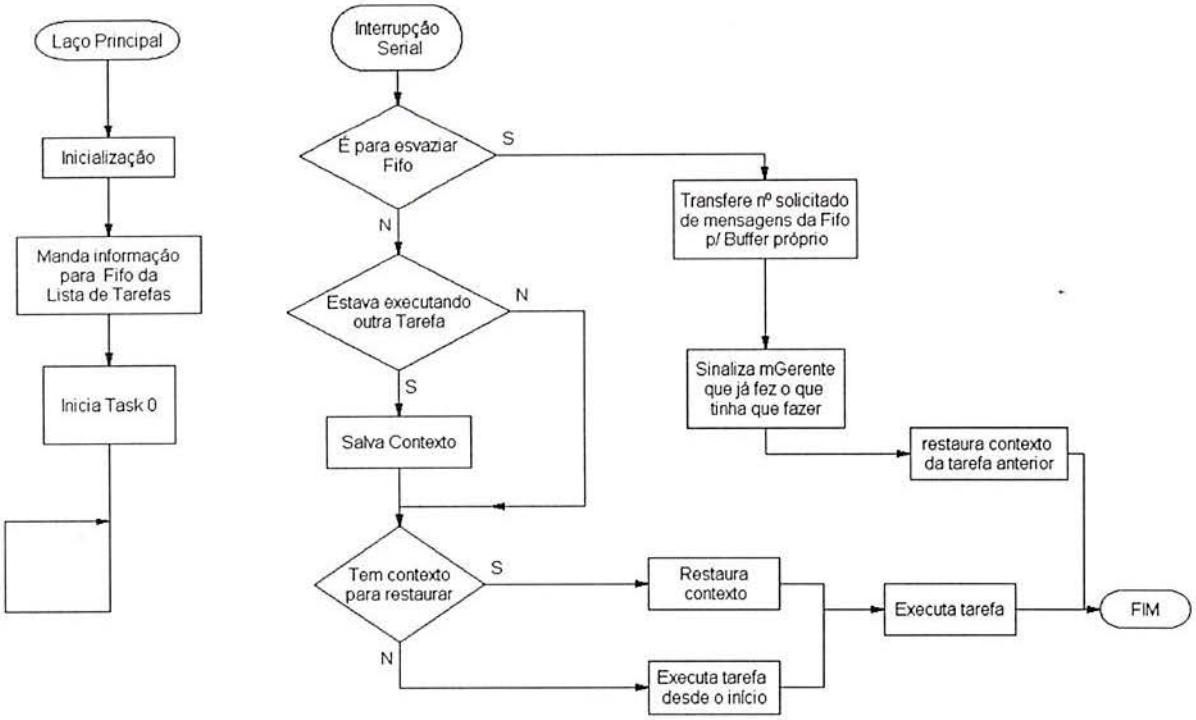


Figura 6-5 Fluxograma do Bloco Executor

6.3.2 Inicialização

Por estar conectada a uma memória RAM não-volátil, com a inicialização da UP, µPrincipal possui como tarefa enviar ao µGerente e ao µComunicação as informações que estes precisam para que possam entrar em funcionamento pleno os seus sistemas operacionais. Estas informações são buscadas na memória supra referida. A existência desta memória faz com que a estrutura das tabelas e as tarefas da aplicação já estejam prontas quando da inicialização da UP. As mesmas só serão modificadas quando for realizado um novo "Download" de uma nova aplicação do usuário.

6.3.3 Recursos:

O µPrincipal utiliza apenas o canal serial, operando no modo 0, que é o responsável pela comunicação com o µGerente. O restante dos recursos disponíveis pelo microcontrolador estão reservados para uso pela aplicação do usuário.

6.3.4 Serviços do Sistema Operacional:

O sistema operacional do µPrincipal disponibiliza os seguintes serviços:

6.3.4.1 Serviços Destinados ao Chaveamento entre tarefas

6.3.4.1.1 Salvamento de contexto

Serviço responsável pelo salvamento do contexto da tarefa atualmente em execução. Sempre antes de ser chamada, devem ter sido salvos os registros A, B, DPL, DPG, PSW e SP em variáveis existentes para este fim específico. Implementado pela função "SaveContext".

6.3.4.1.2 Interrupção serial

Rotina de atendimento de uma interrupção serial que ocorre sempre que o µGerente precisa enviar uma mensagem para o µPrincipal. Esta mensagem é interpretada e as providências necessárias são tomadas pelo µControlador. Implementado pela função "serial_interrupt".

6.3.4.2 Serviços Destinados a preparação prévia para execução das tarefas

6.3.4.2.1 *Download*

Serviço que realiza a transferência da FIFO de recebimento de mensagem para a memória RAM não-volátil, de um novo código para a aplicação do usuário. Isto é realizado sempre que se desejar atualizar ou modificar a aplicação que está sendo executada na arquitetura. Implementado pela função "Download".

6.3.4.2.2 Confecção das tabelas necessárias

Serviço que analisa as tarefas existentes na nova aplicação que a função Download transferiu para a memória RAM e prepara as tabelas existentes para que as tarefas possam ser executadas. Implementado pela função "faz_tabelas".

6.3.4.3 Serviço de espera por uma nova tarefa para executar no µPrincipal

6.3.4.3.1 Começo de execução da aplicação do usuário

Serviço que destina-se a iniciar os programas aplicativos do usuário após um download, ou esperar por uma nova tarefa após alguma outra ter sido suspensa ou cancelada.

Esta função é necessária devido às características de empilhamento de chamadas a funções utilizadas no sistema operacional do µPrincipal. Implementado pela função "Start".

6.3.4.4 Serviço de procura de tarefa na tabela das características das tarefas

6.3.4.4.1 Localização de tarefa na tabela.

Serviço que procura a tarefa na tabela e devolve sua posição relativa em relação ao início da mesma, sendo que a primeira tarefa da tabela possui posição relativa 0. Implementado pela função "cSearchTask".

6.3.4.5 Serviço de troca de mensagens entre as tarefas

6.3.4.5.1 Leitura de mensagens

Serviço responsável por ler uma mensagem armazenada na FIFO Ram e colocá-la no *buffer* apropriado para que possa ser lida pela tarefa a qual ela se destina. Implementado pela função "ReadMsg".

6.3.4.5.2 Envio de mensagens

Serviço responsável pelo envio de uma mensagem para uma tarefa específica. Implementado pela função "SendMsg".

6.3.4.5.3 Retirada de mensagens da FIFO

Serviço que retira da FIFO de recebimento de mensagens uma ou mais mensagens, dependendo do valor especificado pelo µGerente. Implementado pela função "Take_Message_FIFO".

6.3.4.6 Serviços de controle direto sobre a execução de tarefas

6.3.4.6.1 Ativação da tarefa

É o serviço que "ativa" a tarefa passada como parâmetro, que é colocada, no µGerente, na lista de tarefas prontas para execução, com a prioridade especificada. Esta função pode ser utilizada pela tarefa com ID 0 (tarefa primeiramente executada) para ativar outras tarefas, passando o controle do sistema para elas e assim colocando o sistema em operação. Implementado pela função "ActivateTask".

6.3.4.6.2 Cancelamento da Tarefa

Serviço que cancela a programação de uma tarefa determinada, retirando-a de qualquer lista em que possa estar inserida (prontas para a execução, espera de eventos, ...) no µGerente. Isto faz com que uma tarefa possa ter o poder de eliminar outras tarefas, normalmente tarefas subordinadas, bastando para isso conhecer o seu ID. Implementado pela função "CancelTask".

6.3.4.7 Serviços de programação de espera de eventos

6.3.4.7.1 Espera por mensagem

Serviço que avisa ao µGerente que uma determinada tarefa estará bloqueada até a chegada de uma mensagem para a mesma, oriunda de uma tarefa específica. Pode ser reprogramada a prioridade com que a tarefa deve ser ativada quando da chegada da referida mensagem. Se enviar zero como ID da tarefa de origem da mensagem que está esperando, indica que qualquer mensagem que esta tarefa receber é suficiente para desbloqueá-la. Implementado pela função "WaitMsg".

6.3.4.7.2 Programar a espera por eventos assíncronos

Serviço que programa a espera de um evento assíncrono a ocorrer em um dos pinos de entrada para este tipo de evento existentes no µGerente. Junto com a informação do pino relacionado, é informada a prioridade de disparo da tarefa na chegada deste evento. Se a tarefa deve programar esta espera de evento e bloquear, o bit *Suspend* deve ser ligado; se ainda serão programados outros eventos antes de causar a suspensão da tarefa, *Suspend* deve ser zerado. O evento é caracterizado por uma transição de 1 para 0 no pino em questão. Implementado pela função "ProgramAsyncEvent".

6.3.4.7.3 Programar a espera por eventos síncronos

Serviço que programa a espera de um evento síncrono (dependente do horário) de uma tarefa, especificando o dia, hora, minuto, segundo, centésimo e/ou milésimo de segundos. Este evento será programado com uma prioridade agregada. O evento pode ser do tipo síncrono absoluto ou síncrono relativo. Se o evento síncrono relativo for cíclico (é automaticamente reprogramado após o acontecimento do evento) deve-se setar a variável *Cyclic*. Se a tarefa deve programar esta espera de evento e bloquear, o bit *Suspend* deve ser

setado; se ainda serão programados outros eventos antes de causar a suspensão da tarefa, deve ser zerado. No caso de um evento síncrono relativo, os dias são ignorados.

Deve-se notar que no caso de querermos programar uma tarefa como cíclica a partir de outra tarefa, como no exemplo, normalmente não queremos que a tarefa que estamos executando no momento suspenda, diferentemente de programarmos um evento síncrono para a tarefa corrente, onde depois de programado o evento, normalmente queremos que a tarefa suspenda e espere a chegada do evento. Ambos os casos são possíveis com o sistema operacional desenvolvido. Implementado pela função "ProgramSyncEvent".

6.3.5 Tabelas e áreas de memória reservadas

O µPrincipal dispõe de algumas tabelas situadas na memória de dados externa. São elas:

6.3.5.1 Tabela das tarefas.

Contém informações sobre as tarefas da aplicação do usuário que fazem parte desta unidade de processamento (UP). Estas informações são as seguintes:

- ID da tarefa *(1 byte)*
- setor de início da próxima mensagem *(1 byte)*
- PC (*program counter*) inicial da tarefa *(2 bytes)*
- tipo de salvamento de contexto *(1 byte)*
- endereço inicial dos dados do contexto *(2 bytes)*
- tamanho da pilha: número de bytes salvos (default = 20 bytes) *(1 byte)*
- endereço inicial dos dados da pilha *(2 bytes)*

6.3.5.2 Tabela dos contextos das tarefas

Contém as variáveis salvas nos chaveamentos de contexto. A quantidade de informação que necessita ser salva depende de cada tarefa. Há três tipos de salvamento de contexto implementado: o *tipo 2*, que salva os registradores A, B, SP, PSW e DPTR, além do PC; o *tipo 1*, que salva todos os anteriores e mais R0 e R1 do banco 0; e o *tipo 0*, que salva ainda os registradores R2 à R7 do banco 0. Estes três tipos de salvamento de contexto foram implementados para possibilitar um chaveamento de contexto mais eficaz nas tarefas que não

trabalham com muitos registradores internos, e chaveamentos mais completos para as tarefas que necessitam guardar um número maior de informações para que seu processamento possa ser retomado a partir de onde foi interrompido.

6.3.5.3 Área do *buffer* de mensagens

As mensagens que chegam para as tarefas locais são armazenadas nesta área de memória, que por sua vez é dividida em "setores". Uma mensagem pode ocupar um ou mais setor, dependendo do seu tamanho. Cada setor só pode armazenar uma mensagem de cada vez. Os setores são identificados por números conforme sua ordem (posição) na memória, sendo que o primeiro é o setor 1. Se um certo setor for o último de uma série, ou seja, se a mensagem nele contida acabar neste setor e não houver mais mensagens para a mesma tarefa depois desta, no último byte do setor é escrito o código de fim de mensagens (FF_h), senão é escrito o número do setor onde há a continuação da mensagem ou o setor onde está o início da próxima mensagem para a tarefa em questão. Os dois primeiros bytes de cada mensagem são respectivamente o ID da tarefa-origem e o tamanho da mensagem.

A dinâmica de armazenamento das mensagens nestes setores é a de uma lista encadeada e pode ser melhor compreendida se forem analisadas as Figuras 6-6 até 6-10, que mostram uma seqüência de chegada de mensagens, e como estas são armazenadas:

Legenda:
GFFS → Indica lugar para onde está apontando a variável GcFirstFreeSector. Esta variável indica qual o primeiro setor livre para armazenamento de mensagens
Tf X → Indica em qual setor está localizado a primeira mensagem para a tarefa X.
Tf Y → Indica em qual setor está localizado a primeira mensagem para a tarefa Y.

1.Todos os setores estão livres:

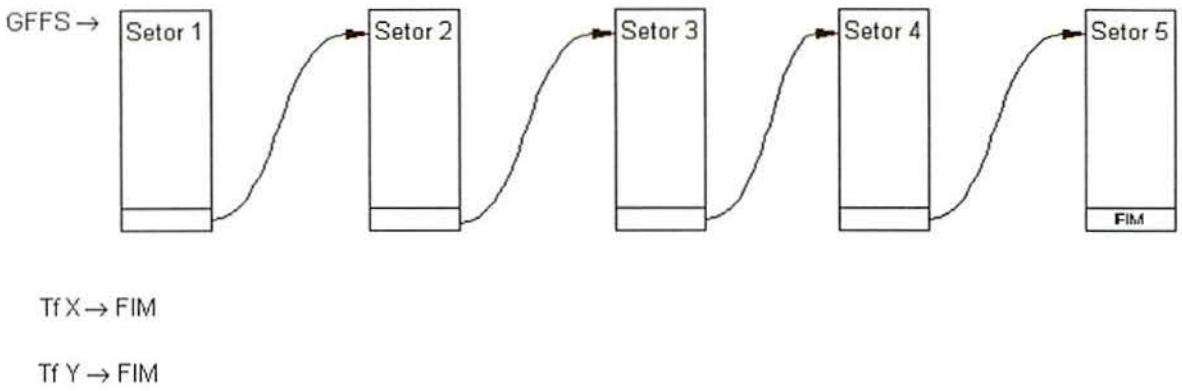


Figura 6-6 Exemplo de armazenamento de mensagens - passo 1

2. Chegou uma mensagem para a tarefa X:

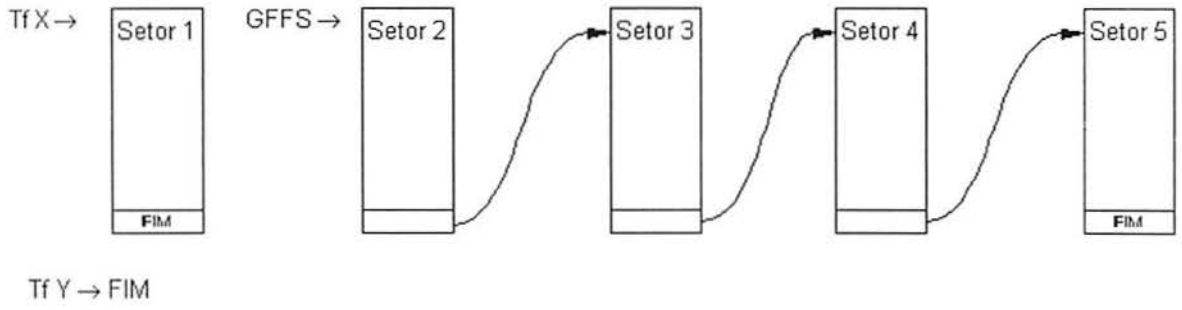


Figura 6-7 Exemplo de armazenamento de mensagens - passo 2

3. Chegou uma mensagem para a tarefa Y:

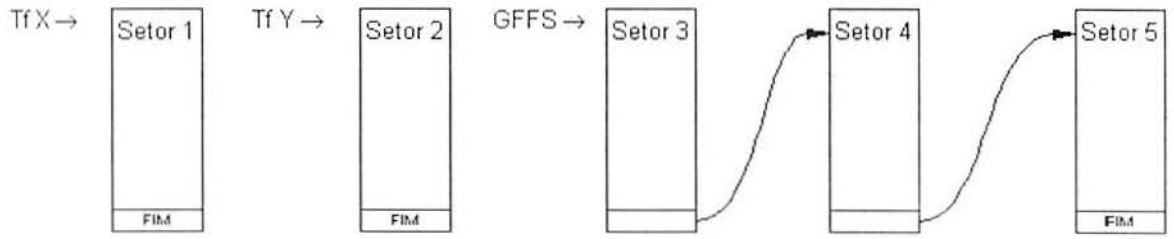


Figura 6-8 Exemplo de armazenamento de mensagens - passo 3

4.A mensagem do setor 1 é lida pela tarefa X:

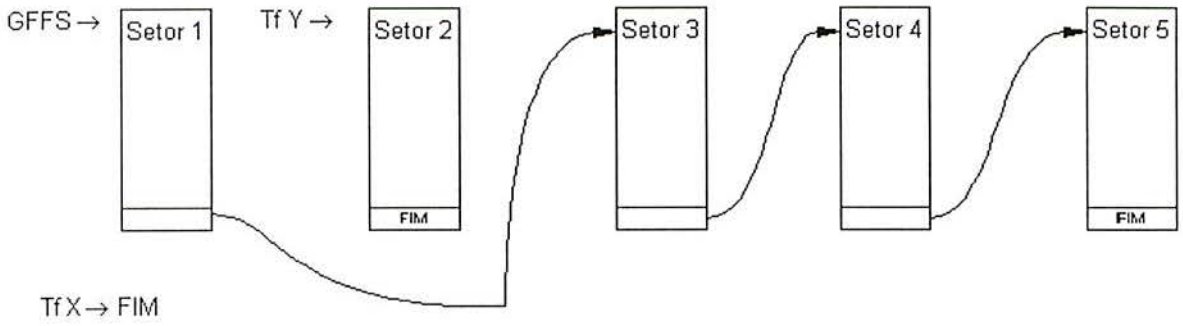


Figura 6-9 Exemplo de armazenamento de mensagens - passo 4

5. Chega uma mensagem que ocupa dois setores para a tarefa Y:

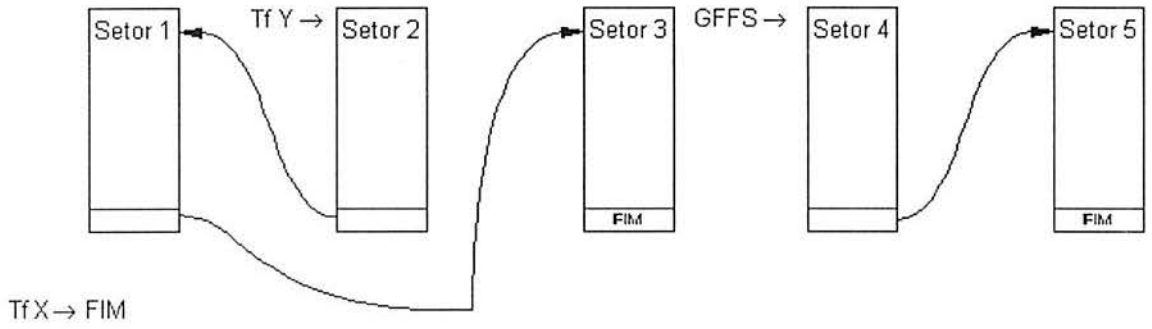


Figura 6-10 Exemplo de armazenamento de mensagens - passo 5

6.3.5.4 Área dos *buffers* de pilhas das tarefas

Contém os dados das pilhas das tarefas salvas durante o chaveamento de contexto. O PC atual é salvo nesta área de dados.

6.3.6 Aplicativos:

Os programas gerados pelo usuário, chamados de aplicativos, não devem utilizar indiscriminadamente a memória interna do µControlador, sob pena de invadir algum espaço de memória utilizado pelo sistema operacional e modificar algum dado importante. Todas as variáveis dos aplicativos devem utilizar a memória externa.

Para a codificação das tarefas, recomenda-se construí-las como funções simples, sem parâmetros de entrada nem parâmetros de retorno. Para troca de informações entre tarefas deve-se utilizar o envio de mensagens. Para cada tarefa, é associado um ID (identificador) que pode variar de 1 a MAX_ID (63 atualmente). Deve ser criada uma tarefa "especial", cujo ID é

0 (zero) e cuja função é a de inicializar o sistema. Esta tarefa é sempre a primeira a ser executada e deve ativar no mínimo uma tarefa antes de acabar, passando o controle para esta quando for terminada. Nenhuma outra tarefa começa a ser executada enquanto esta tarefa não acabar de executar.

6.4 µCOMUNICAÇÃO

O µComunicação tem por função coordenar a troca de mensagens entre as tarefas e a comunicação entre diferentes Unidades de Processamento (UPs).

6.4.1 Inicialização

Quando da inicialização da Unidade de Processamento, recebe do µPrincipal as seguinte informação:

• Código Identificador de todas as tarefas existentes na UP. Com esta informação o µComunicação uma tabela com as tarefas locais a UP, possibilitando que sempre que necessitar gerenciar alguma mensagem saiba se as tarefas origem e destino são locais ou não a sua UP.

O fluxograma da Figura 6-11 mostra como o núcleo do Bloco de Comunicação foi estruturado.

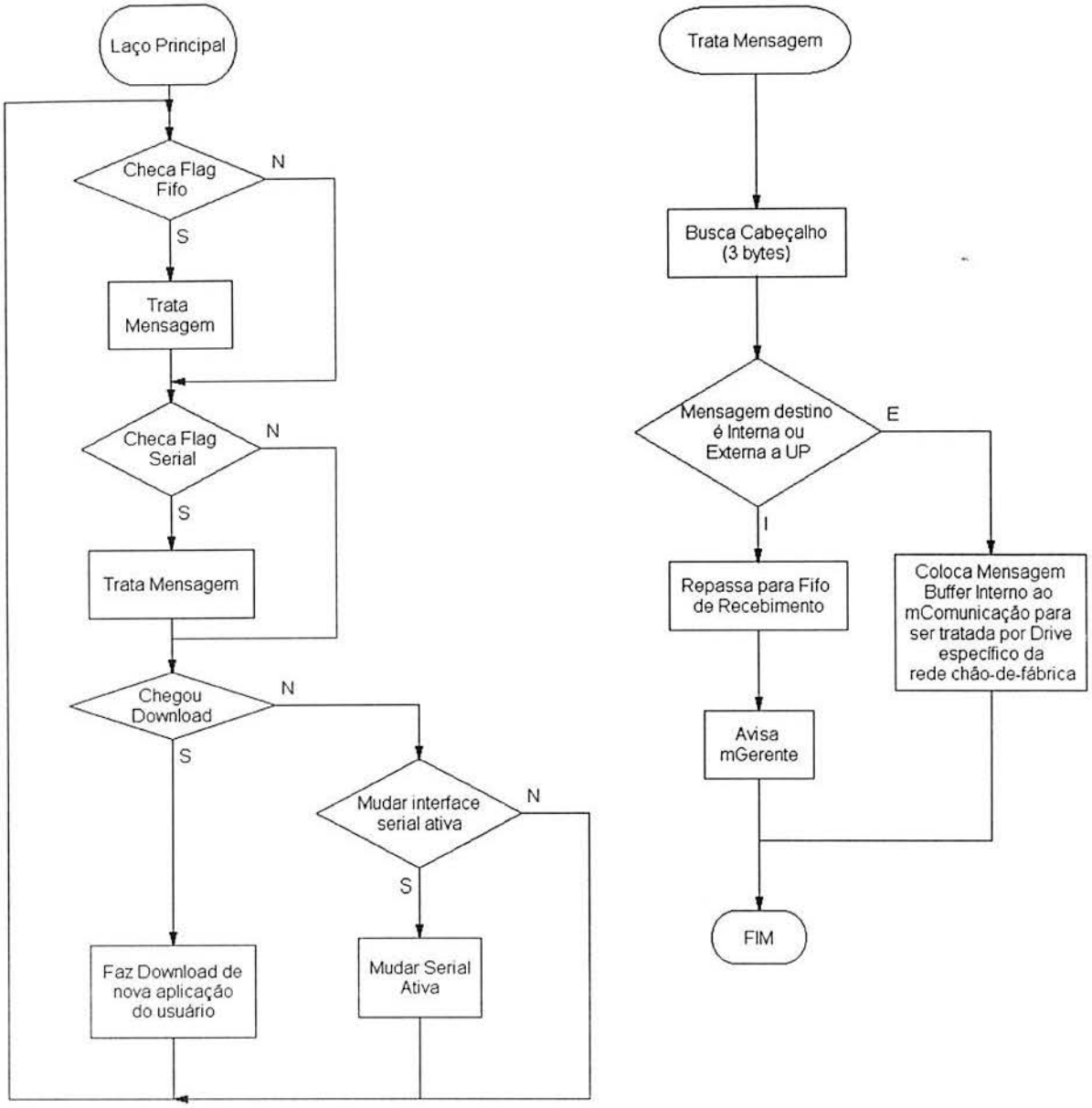


Figura 6-11 Fluxograma do Bloco de Comunicação

6.4.2 Recursos:

O µComunicação utiliza:

timer 1 → temporizar o *time-out* do µGerente na comunicação entre ambos

timer 2 → usado para definir o *baud-rate* da porta serial

interrupção externa 0 → ativada pela FIFO na chegada de dados vindos do µPrincipal

interrupção externa 1 → gerada pelo µGerente na transmissão de um bit para o µComunicação

canal serial → opera no modo 1 e faz a comunicação com outras UPs, através de um barramento chão-de-fábrica (Profibus) ou de um protocolo RS232.

O timer 0 fica disponível para ser utilizado para implementação do protocolo de comunicação existente no barramento industrial a que a UP está conectada.

6.4.3 Serviços do Sistema Operacional

6.4.3.1 *Download* de nova aplicação

Serviço chamado quando vai ser efetuado um download de um novo aplicativo do usuário. Tem por função receber e repassar os dados para os outros µcontroladores, além de criar a sua tabela interna de tarefas presentes na UP correspondente. Implementado pela função "UC_Download".

6.4.3.2 Procura de tarefa na tabela

Serviço que procura uma determinada tarefa na tabela das tarefas presentes nesta UP. Utilizada pelo µComunicação, quando este recebe do µPrincipal uma mensagem entre tarefas, para saber se a tarefa destinatária está nesta UP (mensagem interna) ou em outra UP (mensagem externa). Implementado pela função "bLookForTask".

6.4.3.3 Recebimento de mensagens do µPrincipal

Serviço que trata da interrupção gerada sempre que chegar uma ou mais mensagens na FIFO vindas do µPrincipal. A interrupção é ativa pelo nível, pois garante que haja outra interrupção quando acontecer de outra mensagem chegar enquanto a rotina da anterior ainda estiver sendo tratada. Implementado pela função "external_interrupt_0".

6.4.3.4 Interrupção serial

Serviço que trata da interrupção serial, gerada sempre que houver tráfego de informações pelo barramento industrial ao qual a unidade de processamento está conectada ou quando quisermos iniciar um download (via RS-232) de um novo aplicativo. Implementado pela função "serial_interrupt".

6.4.3.5 Interrupção externa 1

Serviço de tratamento da interrupção externa causada pelo µGerente, sempre que um bit for comunicado. Implementado pela função "external_interrupt_1".

6.4.3.6 Laço principal de execução

Contém, além das inicializações, o laço principal onde são verificados e tratados (ou desviados para as respectivas funções de tratamento) os casos de envio de mensagem para outra UP, chegada de download, chegada de dados de outra UP, envio de mensagem para tarefa local (situada na mesma UP) e mudança de protocolo de RS-485 para RS-232. Implementado pela função "main".

6.4.4 Tabelas

Há apenas uma tabela no μComunicação, onde constam as tarefas (seus IDs) locais, ou seja, as tarefas que executam nesta Unidade de Processamento (UP). Dessa forma, quando uma tarefa desejar enviar uma mensagem para outra, tem-se condições de saber se a tarefa destinatária é local ou externa. Para indicar o fim da tabela, o último elemento tem o valor END_OF_TABLE (0 = valor não válido como ID das tarefas pois é o ID do "main").

6.4.5 Comunicação entre diferentes UPs

A comunicação entre diferentes UPs, responsabilidade do μComunicação, normalmente é executada por intermédio de uma rede chão-de-fábrica (Profibus no caso do *software* desenvolvido). O μComunicação coloca a mensagem a ser enviada em um *buffer* interno para que possa ser enviada pelo *driver* de comunicação ativo no momento. Quando ocorrer a chegada de mensagem da rede chão-de-fábrica para a UP, o *driver* é encarregado de separar as informações pertinentes ao protocolo de comunicação da mensagem propriamente dita, colocando esta em outro *buffer* interno para que possa ser interpretada e tratada pelo μComunicação.

6.5 SINCRONIZAÇÃO ENTRE AS UNIDADES DE PROCESSAMENTO

A sincronização entre as diferentes unidades de processamento é efetuada periodicamente, por iniciativa de uma estação da rede chão-de-fábrica previamente determinada. Em um certo momento, esta estação envia a todas as outras estações o pedido do horário local atual de cada uma das Unidades de Processamento. Cada UP ao receber esta requisição, inicia a contagem de tempo em um timer do seu μComunicação, enquanto é feito o pedido de consulta do horário atual ao μGerente. Uma vez que o μGerente responda ao μComunicação o horário atual do RTC, este pára o timer. O tempo indicado pelo timer é

descontado do horário lido do RTC. Com esta manobra, a UP consegue enviar a estação solicitante um horário bem aproximado do instante em que este foi solicitado.

O horário de cada Unidade de Processamento é comparado pela estação solicitante com o seu próprio horário (que, por definição, é o horário padrão). No caso de haver uma diferença no horário de alguma UP, a estação solicitante envia à mesma uma correção do seu relógio, informando quantos milissegundos devem ser adiantados ou atrasados para que este se aproxime do padrão.

7 RESULTADOS OBTIDOS

Para validação da arquitetura aqui apresentada, foram realizados alguns testes que viabilizaram mensurar o ganho de poder de processamento obtido com esta arquitetura.

Com o objetivo de melhor avaliar os benefícios alcançados com o uso da nova arquitetura, o processo de avaliação foi dividido nas seguintes etapas:

- Avaliação do acréscimo do poder de processamento ocasionado pela inclusão do Bloco Administrador, e conseqüente diminuição da sobrecarga imposta pelas rotinas de gerenciamento de tarefas através de um estudo de caso.

- Avaliação da relação entre a freqüência de ativação das tarefas e o peso computacional do sistema operacional na arquitetura convencional e nova arquitetura proposta.

- Avaliação do acréscimo do poder de processamento ocasionado pela inclusão do Bloco de Comunicação.

7.1 ESTUDO DE CASO

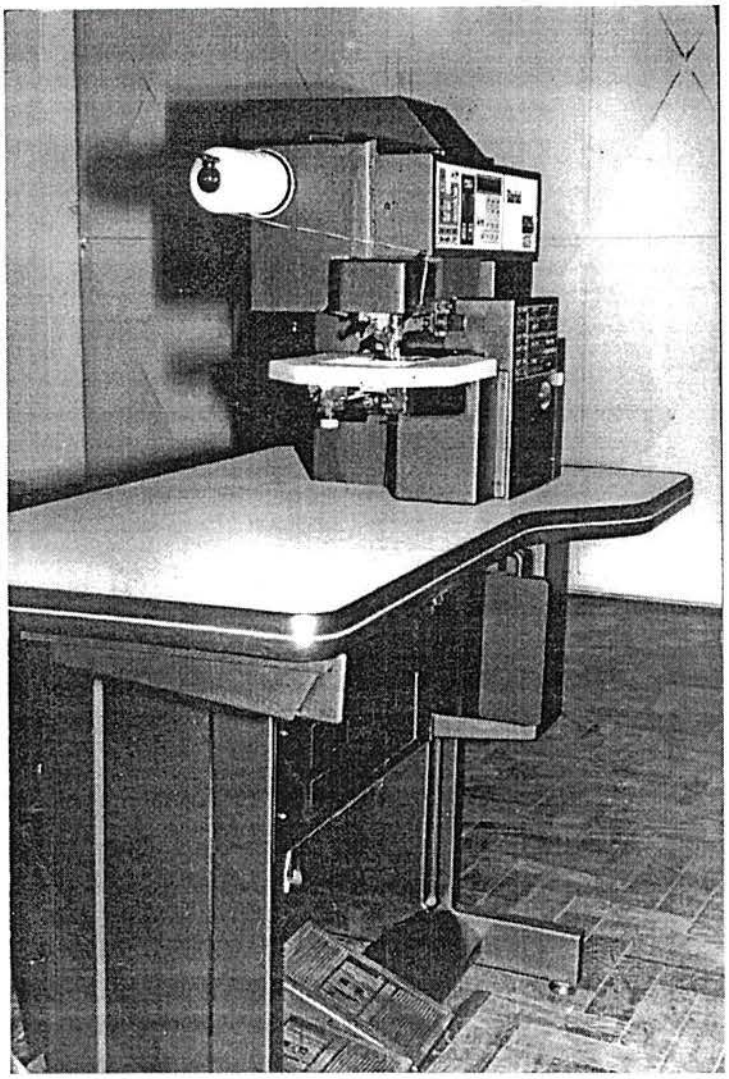


Figura 7-7-1 Máquina de Virar Cortes - fabricado por Máquinas Kehl S/A

Para comparação foi utilizada como base uma aplicação desenvolvida por uma indústria do setor calçadista gaúcho, Máquinas Kehl S/A Indústria e Comércio, que realiza o controle de uma máquina utilizada para a manufatura de calçados, Figura 7-1. Esta aplicação possui antecedentes de busca de atualização tecnológica, visto que após anos de operação, decidiu investir no desenvolvimento de um núcleo tempo-real de sistema operacional multi-tarefa com requisitos de tempo-real executando em uma placa dedicada com um único microcontrolador. O resultado desta transformação foi bem sucedido, com os autores ressaltando a maior naturalidade de implementação de tarefas por processo ao invés de um programa único e a possibilidade de verificação da real ocupação do processador, indicador este que sinaliza para a viabilidade ou não de se acrescentar novas ferramentas ao sistema. A existência deste antecedente, faz com que os proprietários do sistema estejam abertos para

inovações tecnológicas, disponibilizando as informações e documentos necessários para que testes fossem feitos baseando-se no supra referido sistema.

Além disto, a aplicação inclui um razoável número de tarefas concorrentes, incluindo tarefas síncronas e assíncronas. São estas:

- Painel (Prioridade=30): Tarefa que realiza a interface com o operador através do painel. Ativada pelo pressionamento de uma tecla ou por passagem de tempo.
- Tipo_Borda (Prioridade=35): Tarefa que realiza o sensoriamento do tipo de borda da peça. Ativado por evento (presença da peça) ou passagem de tempo.
- Atuação (Prioridade=50): Tarefa que realiza as atuações de cola, picote, redução e ajuste de largura. Ativação síncrona relativa.
- Semi (Prioridade=45): Tarefa que realiza a execução do controle semi-automático de marchas nas bordas. Ativação síncrona relativa.
- Programa (Prioridade=45): Tarefa que grava a programação gestual da execução da peça. Normalmente tarefa cíclica. Quando perceber solicitação de programação, passa a ser tarefa assíncrona até cumprir todas etapas de programação, quando então volta a ser tarefa cíclica.
- Auto (Prioridade=90): Tarefa que executa a programação gestual da peça. Normalmente tarefa cíclica. Quando perceber solicitação de programação, passa a ser tarefa assíncrona ou síncrona relativa, até cumprir todas etapas de programação, quando então volta a ser tarefa cíclica.
- Temp_Bico (Prioridade=20): Tarefa que modula o acionamento das resistências de reaquecimento do conduto de cola até o bico de aplicação. Ativação síncrona relativa.
- Temp_Tnq_Mede (Prioridade=25): Tarefa que adquire a temperatura da cola no tanque, converte para índice e exibe em *bargraph*, compara com o nível ajustado e gera os indicadores de controle da modulação. Tarefa cíclica.
- Temp_Tnq_Mod (Prioridade=5): Tarefa que modula as resistências de reaquecimento do tanque de cola, de acordo com as medições e ajustes realizados. Ativação síncrona relativa.

- Mot_Ativo (Prioridade=10): Tarefa que controla a ativação do auto-desligamento do motor da máquina, quando o mesmo alcançar 4 minutos sem uso. Tarefa Cíclica.
- Sol_Prot (Prioridade=15): Tarefa que simula a temperatura dos solenóides, ativa indicadores de desligamento de segurança e exibe situação no display. Tarefa cíclica.

Como forma de obter uma comparação entre a nova arquitetura e uma arquitetura convencional foram compiladas duas versões da aplicação destinadas a serem executadas no mesmo *hardware*: A primeira versão foi implementada somente com o uso do Bloco Executor, enquanto a segunda foi distribuída entre o Bloco Executor e o Bloco Administrador. Com esta providência, a influência de eventuais discrepâncias nos resultados causadas por diferentes ciclos de máquina, freqüência do relógio e outros detalhes específicos de cada *hardware* pôde ser evitada.

O sistema operacional original da aplicação inclui funções para criação e ativação de tarefas, bem como funções para modificar o estado da tarefa atualmente em execução. Existem três filas a serem gerenciadas pelo sistema: uma para as tarefas prontas para execução, uma para todas as tarefas declaradas e outra para as tarefas que estão esperando pela chegada de algum evento, quer seja ele síncrono ou assíncrono. O algoritmo de escalonamento original utilizado é o de prioridade fixa.

O núcleo original possui um processo cíclico, ativado a cada 25 ms, responsável pelo *polling* dos eventos assíncronos e pela ativação das tarefas com requisitos temporais. Em entrevista com os desenvolvedores do sistema, estes relataram que o valor de 25 ms foi obtido de forma empírica, levando em conta o custo-benefício entre o tempo de latência e a sobrecarga causada no processador. É importante salientar que este núcleo de sistema operacional para aplicações tempo-real é um sucessor de um sistema operacional não multi-tarefa que tem sido utilizado pela empresa por diversos anos. Os desenvolvedores do sistema relataram que já estão bastante entusiasmados com as vantagens obtidas pelo uso de um sistema operacional multi-tarefa, pois acham mais fácil e natural a implementação de aplicações com o uso de tarefas concorrentes, além do sistema operacional multi-tarefa ter permitido a obtenção da real ocupação do processador. Esta informação foi conseguida pela definição de uma tarefa, com a prioridade mais baixa possível, que está sempre pronta para execução. Como conseqüência, esta só estará sendo executada quando não existir nenhuma

outra tarefa pronta para execução. Pela medição do percentual de tempo que o processador está executando esta tarefa, pode-se ter a informação do percentual de tempo que o processador tem disponível para a inclusão de novas tarefas a serem processadas pela UP.

7.2 AVALIAÇÃO DO ACRÉSCIMO DO PODER DE PROCESSAMENTO OCASIONADO PELA INCLUSÃO DO BLOCO ADMINISTRADOR

A primeira avaliação realizada foi a comparação do ganho de processamento obtido com a inclusão do Bloco Administrador no sistema. Isto resultou na retirada do algoritmo de escalonamento e da gerência dos eventos síncronos e assíncronos da concorrência pelo microcontrolador com as tarefas do usuário. Obviamente, tal ganho será diretamente proporcional ao número de eventos assíncronos, ao número de tarefas dependentes de requisitos temporais, bem como à complexidade do algoritmo de escalonamento das tarefas.

Os resultados de laboratório obtidos com a utilização da arquitetura proposta nesta dissertação foram bem expressivos. Uma das principais vantagens obtidas com a nova arquitetura foi a diminuição do período de *polling*, dos originais 25 ms para 1 ms, sem a presença dos efeitos colaterais de sobrecarga do sistema que aconteceria com a arquitetura original. A importância deste fato pode ser percebida pelo seguinte exemplo: Considerando duas tarefas com requisitos de ativação assíncronos, como por exemplo uma interrupção especial de teclado e a chegada de um novo sapato para ser processado na esteira. Ambas as tarefas tem um tempo computacional pequeno, e a tarefa de atendimento à chegada de um novo sapato possui uma prioridade maior. Como descrito na figura 7-2, na nova arquitetura ambos os sinais podem ser reconhecidos e as tarefas concluídas antes que a versão antiga conseguisse ao menos detectar a presença dos sinais. Outro efeito da diminuição do tempo de *polling* é que, no exemplo, as tarefas foram executadas na ordem em que os eventos aconteceram, o que não ocorreu na arquitetura original. Se pensarmos que a tecla pressionada seria para a modificação da forma como a borda do sapato será confeccionada, no exemplo dado, teríamos, na arquitetura original, uma unidade a mais produzida da forma antiga. Já na nova arquitetura, o próximo sapato presente seria confeccionado já com a nova programação.

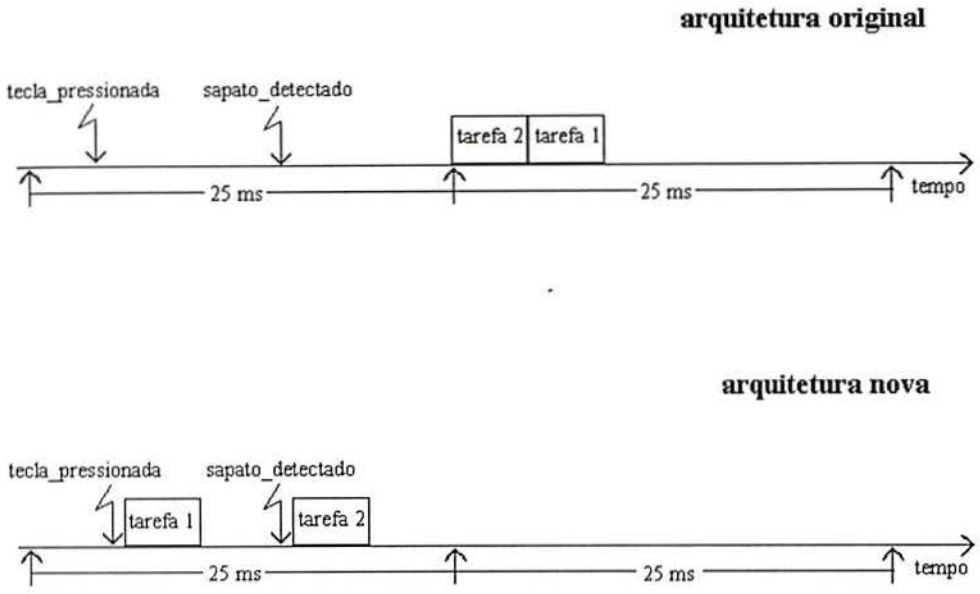


Figura 7-2 Comparação de resposta entre a arquitetura tradicional e a nova

Outro importante aspecto foi a comparação entre a sobrecarga média causada pelo sistema operacional original e a obtida com a utilização do sistema operacional desenvolvido para a nova arquitetura. Como mostrado na Figura 7-3, quando executadas no mesmo processador das tarefas da aplicação, as funções do sistema operacional consomem em torno de 25% do tempo de processamento da UP. Na nova implementação sugerida, este tempo não ultrapassa 1,5%. É claro que somando-se o tempo de execução de ambos os processadores, µPrincipal e µGerente, obtêm-se aproximadamente o mesmo resultado do caso com um processador somente. Na verdade, o tempo de execução total das funções do sistema operacional é um pouco superior na nova arquitetura, uma vez que existe um tempo dedicado a troca de informações entre os dois µControladores e também devido ao *polling* estar sendo efetuado vinte cinco vezes mais rapidamente que na versão antiga. Contudo, como as funções do sistema operacional não competem mais pelo processador com as tarefas da aplicação, chegamos a esta expressiva diminuição do tempo em que a UP não pode processar tarefas da aplicação devido a estar executando funções do núcleo do sistema operacional. Este fato não

somente conduz a um melhor aproveitamento da UP como também aumenta a previsibilidade quanto ao processamento das tarefas da aplicação, uma vez que a sobrecarga causada pelo sistema operacional é praticamente constante durante toda a execução.

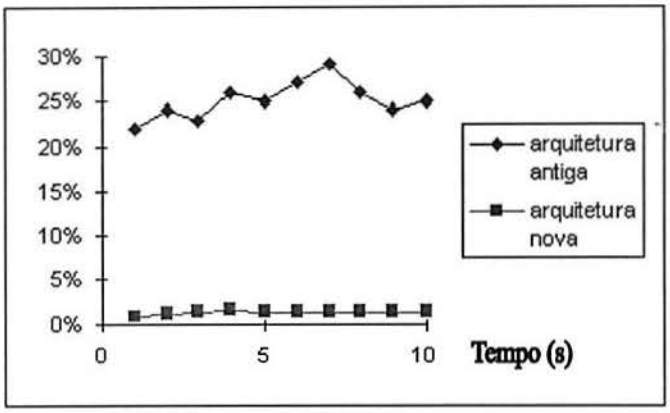


Figura 7-3 Comparação de desempenho entre a arquitetura tradicional e a nova

Como mostrado na Figura 7-4, este 1,4% médio que a nova arquitetura necessita consumir do poder de processamento do µPrincipal são distribuídos nas seguintes funções:

- Comunicação entre o Bloco Executor e o Bloco Administrador (0,40%).
- Programação dos requisitos síncronos de ativação das tarefas (0,22%).
- Programação dos requisitos assíncronos de ativação das tarefas (0.24%).
- Chaveamento de contexto (0,42%).
- Procura pela nova tarefa a executar na fila de tarefas existente na memória Ram externa (0,12%).

Com o objetivo de verificar o quanto estes números poderiam ser modificados por diferentes aplicações, foram feitas algumas análises com o aumento do número de tarefas existentes, o aumento do tempo de processamento de cada tarefa e a mudança de algoritmos de escalonamento. Estes testes conduziram às seguintes conclusões:

- O gasto de tempo para o chaveamento de contexto das tarefas e para comunicação entre o µPrincipal e o µGerente está diretamente relacionado com a densidade de tarefas a serem ativadas em uma unidade de tempo. Isto conduz à conclusão de que estes tempos também são dependentes da política adotada para o algoritmo de escalonamento, sendo maior para aqueles algoritmos que utilizam preempção de tarefas, como mostrado na Figura 7-4.

• O tempo gasto com a programação de requisitos síncronos e assíncronos depende exclusivamente da natureza das tarefas da aplicação e não é influenciado pela política adotada para o algoritmo de escalonamento.

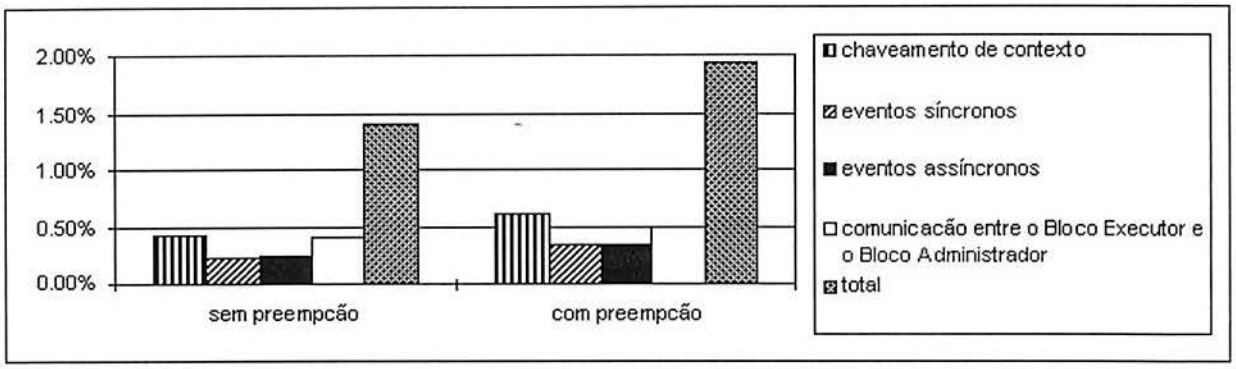


Figura 7-4 Discriminação do consumo com o sistema operacional pelo Bloco Executor

É interessante observar que a utilização de um algoritmo de escalonamento que produz uma sobrecarga menor no sistema, não necessariamente levará a uma melhor performance do sistema como um todo. Por exemplo, na aplicação da máquina para manufatura de calçado, embora com um algoritmo de escalonamento preemptivo a sobrecarga do sistema operacional tenha aumentado de 1,4% para 1,9%, como mostrado na Figura 7-4, este normalmente gera um escalonamento de tarefas mais efetivo. A razão disto é que com o algoritmo preemptivo, fica possível às tarefas com alta prioridade terem acesso ao processador mais rapidamente, reduzindo o tempo de reação à chegada do evento.

7.3 AVALIAÇÃO DA VARIAÇÃO DE COMPORTAMENTO DAS ARQUITETURAS EM ESTUDO COM A VARIAÇÃO DO CICLO DE ATIVAÇÃO DE UM CONJUNTO DE TAREFAS

Com o objetivo de ter um maior conhecimento sobre o comportamento e vantagens da arquitetura proposta sobre as convencionais, realizaram-se dois testes distintos para cada arquitetura com um conjunto de tarefas específicas. Cada um destes testes diferenciava-se do outro pelo período de ativação das tarefas cíclicas. Os testes com períodos de ativação das tarefas cíclicas iguais, não tiveram tempos de emulação exatamente iguais na arquitetura convencional e na nova arquitetura por dificuldades técnicas com os equipamentos utilizados para aquisição de dados. Por este motivo os dados apresentados serão sempre na forma percentual.

7.3.1 Características comuns aos quatro testes (dois para cada arquitetura)

As características comuns aos quatro testes realizados foram:

- Executado o p*olling* dos pinos externos a cada 1 milissegundo.

- Foram utilizadas ao todo no ensaio oito tarefas. Foram escolhidas tarefas cíclicas e assíncronas. As tarefas síncronas absolutas e as tarefas síncronas relativas não foram testadas pois dificultariam a aquisição de dados e não acrescentariam nada de diferente aos resultados. Isto acontece pois as tarefas cíclicas em cada instância são tratadas como se fossem síncronas absolutas pelo sistema operacional que transforma o tempo que falta para a próxima instância da tarefa cíclica ser ativada em um horário absoluto para que a mesma possa ser posta na SIFO. O mesmo acontece com as tarefas síncronas relativas.

- O ciclo de ativação de cada uma das tarefas serão explicitados no detalhamento de cada testes. Os dados comuns para as tarefas em todos os testes são os seguintes:

  - Tarefa 1: tarefa cíclica, prioridade 6.

  - Tarefa 4: tarefa cíclica, prioridade 7.

  - Tarefa 5: tarefa cíclica, prioridade 7.

  - Tarefa 6: tarefa cíclica, prioridade 11.

  - Tarefa 2: foi ativada durante a inicialização e entrou em um laço infinito, de modo a estar sempre pronta para execução com prioridade 3.

  - Tarefa 3: tarefa que programava uma espera por evento em pino externo (assíncrono) com prioridade 9.

  - Tarefa 7: enviava uma mensagem com conteúdo de quatro bytes à tarefa 8, logo após suspendia esperando uma mensagem da tarefa 8 com prioridade 4.

  - Tarefa 8: programava a espera de uma mensagem da tarefa 7 com prioridade 13, cuja chegada ocasionava o envio de uma mensagem de dois bytes à tarefa 7.

- Foi forçado um único evento assíncrono para a tarefa 3 durante o ensaio.

- O algoritmo de escalonamento utilizado em ambos os casos foi o de prioridade fixa.

• Sempre que possível, foram utilizadas as mesmas rotinas para executar atividades semelhantes (por exemplo: rotina de procura de tarefas em alguma tabela, rotina de salvamento de contexto, ...) em ambas as arquiteturas.

• O emulador, necessário para facilitar a aquisição dos dados, foi utilizado como único µcontrolador na arquitetura convencional e como µPrincipal na arquitetura proposta.

• Nas duas arquiteturas uma tarefa síncrona era programada no RTC somente se o seu momento de ativação excedesse 5ms a partir do momento de ativação da última tarefa síncrona a ter sido promovida a pronta. Caso contrário, a tarefa era diretamente promovida a pronta.

Os testes a seguir variam no tempo de emulação, tanto nos teste com freqüência de ativação das tarefas baixa quanto nos teste com freqüência alta por dificuldade técnica para aquisição dos dados. A influência deste fato é minimizada quando a análise é feita considerando os resultados percentuais e não os absolutos.

7.3.2 Testes com a arquitetura convencional

• Nos resultados apresentados a seguir, não está explicitado o custo computacional decorrentes da utilização de um Driver para Comunicação com a Rede Chão-de-Fábrica (DCRCF). Este será analisado em separado na seção 7.4.

7.3.2.1 Teste com freqüência baixa de ativação das tarefa

• Ciclos de ativação das tarefas cíclicas:

  • Tarefa 1: 1,0 s

  • Tarefa 4: 0,5 s

  • Tarefa 5: 1,25 s

  • Tarefa 6: 2,5 s

• Tempo de emulação: 5,22 s

- Dispêndio com as funções do sistema operacional:

	(%)
Chaveamento de contexto:	0,429
Tratamento de eventos:	0,771
Programação de evento síncrono:	0,114
Programação de evento assíncrono:	0,044
Programação de ativação de tarefa:	0,065
Programação de espera de mensagem:	0,082
Envio de mensagem:	0,044
Leitura de mensagem:	0,042
***Polling* do evento externo:**	9,600
Espera de nova tarefa:	0,000
TOTAL	11,191 +DCRCF

7.3.2.2 Teste com freqüência alta de ativação das tarefa

- Ciclos de ativação das tarefas cíclicas:
  - Tarefa 1: 0,11 s
  - Tarefa 4: 0,06 s
  - Tarefa 5: 0,15 s
  - Tarefa 6: 0,27 s
- Tempo de emulação: 1,48 s

- Dispêndio com as funções do sistema operacional:

	(%)
Chaveamento de contexto:	5,578
Tratamento de eventos:	8,305
Programação de evento síncrono:	0,537
Programação de evento assíncrono:	0,174
Programação de ativação de tarefa:	0,246
Programação de espera de mensagem:	0,289
Envio de mensagem:	0,160
Leitura de mensagem:	0,149
***Polling* do evento externo:**	9,600
Espera de nova tarefa:	0,000
TOTAL	25.038 +DCRCF

Cabe destacar que destes testes com a arquitetura convencional, certos detalhes já esperados se confirmaram, quais sejam:

- Com o aumento da freqüência de ativação das tarefas, o percentual necessário para tratamento dos eventos, pelo sistema operacional aumentou consideravelmente, pois estes acontecem mais freqüentemente.
- O dispêndio percentual com o *polling* de eventos externos ficou constante, uma vez que não dependem da freqüência de ativação das tarefas.

7.3.3 Testes com a nova arquitetura

7.3.3.1 Teste com freqüência baixa de ativação das tarefa

- Ciclos de ativação das tarefas cíclicas:
  - Tarefa 1: 1,0 s
  - Tarefa 4: 0,5 s

- Tarefa 5: 1,25 s
- Tarefa 6: 2,5 s

- Tempo de emulação: 5,44 s
- Dispêndio com as funções do sistema operacional:

	(%)
Chaveamento de contexto:	0,760
Tratamento de eventos:	0,000
Programação de evento síncrono:	0,235
Programação de evento assíncrono:	0,078
Programação de ativação de tarefa:	0,154
Programação de espera de mensagem:	0,010
Envio de mensagem:	0,099
Leitura de mensagem:	0,031
***Polling* do evento externo:**	0,000
Espera de nova tarefa:	0,126
TOTAL	1,492

7.3.3.2 Teste com freqüência alta de ativação das tarefa

- Ciclos de ativação das tarefas cíclicas:
  - Tarefa 1: 0,11 s
  - Tarefa 4: 0,06 s
  - Tarefa 5: 0,15 s
  - Tarefa 6: 0,27 s
- Tempo de emulação: 1,91 s

- Dispêndio com as funções do sistema operacional:

	(%)
Chaveamento de contexto:	4,773
Tratamento de eventos:	0,000
Programação de evento síncrono:	0,674
Programação de evento assíncrono:	0,222
Programação de ativação de tarefa:	0,468
Programação de espera de mensagem:	0,278
Envio de mensagem:	0,027
Leitura de mensagem:	0,087
***Polling* do evento externo:**	0,000
Espera de nova tarefa:	0,888
TOTAL	7,417

7.3.4 Análise dos resultados obtidos

Tabela Comparativa:

Arquitetura:	Convencional		Proposta		Relação Proposta / Convenc.	
Freqüência de Ativação :	**Baixa (%)**	**Alta (%)**	**Baixa (%)**	**Alta (%)**	**Baixa**	**Alta**
Chaveamento de contexto:	0,429	5,578	0,760	4,773	1,77	0,86
Tratamento de eventos:	0,771	8,305	0,000	0,000	0,00	0,00
Programação de evento síncrono:	0,114	0,537	0,235	0,674	2,06	1,26
Programação de evento assíncrono:	0,044	0,174	0,078	0,222	1,77	1,28
Programação de ativação de tarefa:	0,065	0,246	0,154	0,468	2,37	1,90
Programação de espera de mensagem:	0,082	0,289	0,010	0,278	0,12	0,93
Envio de mensagem:	0,044	0,160	0,099	0,027	2,25	0,17
Leitura de mensagem:	0,042	0,149	0,031	0,087	0,74	0,58
***Polling* do evento externo:**	9,600	9,600	0,000	0,000	0,00	0,00
Espera de nova tarefa:	0,000	0,000	0,126	0,888		
TOTAL	11,191 +DCRCF	25.038 +DCRCF	1,492	7,417	0,13 +DCRCF	0,30 +DCRCF
TOTAL sem considerar *Polling* e Driver para Comunicação com Rede Chão-de-Fábrica	1,591	15,438	1,492	7,417	0,94	0,48

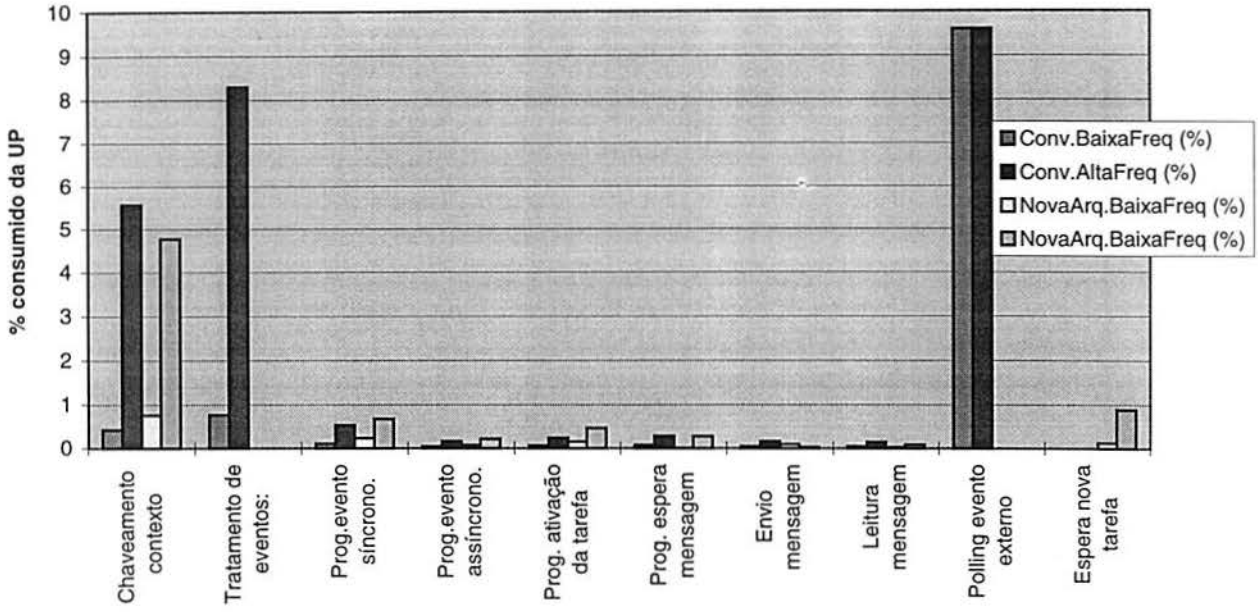


Figura 7-5 Gráfico para comparação dos resultados obtidos

Dos resultados acima expostos, pode-se concluir que as vantagens da nova arquitetura proposta será maior, quanto maior for a freqüência de ativação das tarefas.

Os dois principais fatores que contribuem para este resultado são:

• A não influência das funções de Tratamento de Eventos na nova arquitetura. Funções estas que incluem o algoritmo de escalonamento e a reordenação das diversas filas que o sistema operacional necessita gerenciar. Cabe ressaltar que os resultados acima foram conseguidos com a utilização do algoritmo de escalonamento de Prioridade Fixa, que vem a ser dos mais simples existentes. Se a opção recair por um algoritmo de escalonamento mais complexo, como já explicado em capítulos anteriores, as vantagens da nova arquitetura tendem a aumentar ainda mais.

• O *polling* dos eventos assíncronos precisará ser mais freqüente quanto menor for o tempo de resposta necessário. Como pode ser percebido nas tabelas acima, seu peso computacional é importante e diretamente proporcional a sua freqüência. Se nos testes acima ao invés de 1 KHz tivesse sido utilizado 100 Hz como freqüência de verificação da ocorrência de um novo evento, seu peso computacional baixaria de 9,6% para 0,96%. Se por outro lado, a freqüência de verificação escolhida fosse de 10 KHz, seu peso computacional subiria para 96%, isto é, praticamente seria a única atividade do microprocessador.

7.4 Custo Computacional do Driver de Comunicação com a Rede Chão-de-Fábrica

O custo computacional do driver de comunicação com a rede chão-de-fábrica depende de diversas variáveis, sendo que cada escolha conduzirá a resultados completamente diferentes. As principais variáveis que influenciam nesta variação são:

- Protocolo escolhido para a rede chão-de-fábrica
- Em protocolos mestre-escravo, qual das duas funções for a exercida pela UP
- Taxa de comunicação da rede chão-de-fábrica
- Número de estações conectadas a rede.
- Tamanho médio das mensagens enviadas pela rede.
- Taxa de ocupação da rede.

Estas e outras variáveis influenciarão diretamente na freqüência em que a UP estará envolvida com alguma atividade relacionada a rede chão-de-fábrica.

7.4.1 Estudo de Caso

Como estudo de caso para exemplificar a influência do atendimento dos requisitos de uma rede chão-de-fábrica, serão aproveitados dados coletados para a dissertação de mestrado de Míriam Noemi Cáceres, em andamento no grupo de Instrumentação Eletro-Eletrônica do PPGEM da UFRGS.

- Protocolo: Profibus
- Característica da estação analisada: Escrava
- Velocidade de Comunicação: 28800 bps
- Número de estações da rede: 12 estações escravas e 1 mestre
- Tamanho das mensagens transmitidas: 5 bytes de dados
- Estação mestre: Pentium de 166 MHz, com 16MB de RAM
- Estações escravas: terminais dotados de microcontrolador da família do microcontrolador 8051, elaborados no projeto "Votador Eletrônico" desenvolvido no laboratório de Instrumentação Eletro-Eletrônica da UFRGS.

- Resultados obtidos:

  - Percentual de ocupação da rede: 43,45%

  - A interrupção serial ativa quando cada byte de informação é recebido da rede pelo canal serial do microcontrolador, necessita , no mais rápido dos casos, 42 ciclos de máquina. Para que um byte de informação seja enviado, são necessários 11 bits, pois acrescenta-se ao byte, 1 bit de início da informação, 1 bit de paridade e 1 bit de fim da informação. Para a taxa de transmissão de 28800 bps e cristal do µControlador da estação escrava de 8 MHz (freqüência do cristal utilizado em todos os testes descritos neste capítulo), chegamos que a cada segundo o µControlador do terminal despende 71,67 ms para atender a interrupção serial que é ativa a cada byte recebido da rede chão-de-fábrica. Este valor decorre da fórmula abaixo:

$$\%IntSerial = \frac{TaxaTransmissao * TaxaOcupacao * NumCiclosMaquina * PeriodoCicloMaquina * 100}{NumeroDeBitsPorByteDeInformacao}$$

$$\%IntSerial = \frac{28800 * 0{,}4345 * 42 * 0{,}0000015 * 100}{11}$$

$$\%IntSerial = 7{,}167\%$$

  - Cada vez que um byte de informação é recebido da rede chão-de-fábrica pelo µControlador, este é posto em um buffer específico, para que de tempos em tempos o µControlador verifique se existe mais alguma informação disponível, e a trate convenientemente. O período de verificação arbitrado no teste acima foi de 1ms. No mais rápido dos casos, esta verificação disponde 52 ciclos de máquina. Para um cristal de 8MHz chegamos a um percentual de utilização do µControlador para esta função de pelo menos 7,8 %.

  - O custo computacional total do µControlador com a Rede Chão-de-Fábrica será pelo menos de:

    - %Total = %IntSerial + *%Polling*
    - %Total = 7,167 % + 7,8%
    - %Total = 14,167 %

Estes resultados por si só já demonstram a importância de uma Unidade de Processamento integrada a uma rede de comunicação Chão-de-Fábrica possuir um µControlador dedicado as tarefas de comunicação da unidade. Se ainda considerarmos que redes modernas tendem a se comunicar com taxas de comunicação cada vez mais altas, passando de 1 MHz em alguns casos, a retirada das funções de gerenciamento das tarefas de comunicação do µControlador responsável pelo processamento das tarefas da aplicação passa a ser praticamente obrigatório.

Cabe salientar que os percentuais acima foram obtidos considerando sempre o caso em que menor influência o protocolo teria no desempenho da UP. Portanto, os resultados práticos tendem a ser ainda mais favoráveis a justificar a adoção de um µControlador

específico para a comunicação com a rede chão-de-fábrica, como sugerido na arquitetura proposta nesta dissertação.

Além da perda de poder de processamento pela UP, estes *pollings* necessários causam ainda o efeito de retardar o término da execução de cada tarefa, pois periodicamente o processamento das mesmas é interrompido para que a UP atenda as tarefas nesta seção descritas. Este retardo pode algumas vezes significar a diferença entre o cumprimento ou não dos *Deadlines* das tarefas.

8 CONCLUSÕES E TRABALHOS FUTUROS

Uma arquitetura de baixo custo para o desenvolvimento de sistemas tempo-real distribuídos, que busca resolver alguns dos principais problemas detectados nas arquiteturas convencionais baseadas em um único processador por Unidade de Processamento, foi apresentada nesta dissertação.

A arquitetura faz uso de uma unidade de *hardware* dedicada baseada em microcontroladores de baixo custo. Possui três blocos:

- O Bloco Executor é o responsável pela execução das tarefas da aplicação do usuário.
- O Bloco Administrador é o responsável pelo escalonamento das tarefas e pela gerência dos instantes de ativação destas, quer sejam elas síncronas ou assíncronas. Uma vez que nesta arquitetura o algoritmo de escalonamento das tarefas não mais concorre pelo processador com as tarefas da aplicação, ele pode ser mais sofisticado e especializado. Isto conduz a um sistema que tende a ser mais determinístico, característica importante para o desenvolvimento de aplicações de tempo-real.
- O Bloco de Comunicação é o responsável por todas as atividades relacionadas a comunicação entre tarefas, incluindo a execução do *driver* de comunicação com a rede chão-de-fábrica. É através deste bloco que informações são passadas e recebidas pela rede de comunicação.

Com estas funções não mais sendo feitas pelo processador principal, este possuirá mais tempo para se dedicar a processar as aplicações do usuário. Em um sistema distribuído, temos "n" Unidades de Processamento alocadas para que as tarefas necessárias sejam realizadas. Como cada UP agora pode processar mais tarefas da aplicação, percebe-se que menos UPs serão necessárias para a implantação do sistema.

Embora o custo total de fabricação tenha subido alguns reais devido a inclusão de mais alguns componentes, quando comparado a uma arquitetura convencional, esta elevação do custo com relação ao preço de venda dos equipamentos para uso em sistemas tempo-real distribuídos é proporcionalmente bem menor do que o ganho de processamento que se obteve. Como exemplo de preço de venda pode ser citado os Controladores Lógicos Programáveis (CLP) que são, na data que esta dissertação foi escrita, vendidos por algumas centenas de reais. Cabe ressaltar que os CLPs são estruturalmente ainda mais simples que as arquiteturas que trata esta dissertação, visto que estas são multi-tarefas e os CLPs não.

Os testes confirmaram as expectativas e mostraram que um ganho considerável de processamento pode ser obtido com a utilização da arquitetura proposta, principalmente quando a UP se destina a processar tarefas da aplicação com ciclos de ativação curtos.

Dentre as principais vantagens do sistema proposto destacam-se:

- Aumento do poder de processamento de uma Unidade de Processamento.

- Maior facilidade em obter o determinismo temporal, característica fundamentai em sistemas tempo-real distribuídos.

- . Possibilidade de utilização de algoritmos de escalonamento mais complexos e especializados, sem uma sobrecarga proibitiva no desempenho do sistema.

Como seqüência do trabalho apresentado nesta dissertação, tem-se como metas futuras:

- Melhorar as limitações existentes nos sistemas operacionais desenvolvidos.

- Implementar um ambiente de programação e compilação que possibilite o desenvolvimento de aplicações em uma linguagem de alto nível pelo usuário.

- Estender o núcleo do sistema operacional desenvolvido para suportar o conceito de objetos ativos, como descrito em [PFDH96].

- Prover o *hardware* e o *software* desenvolvidos de documentação suficiente para que possam ser utilizados, no laboratório de Automação do Departamento de Engenharia Elétrica da UFRGS, para o ensino de sistemas tempo-real distribuídos para os cursos de graduação e pós-graduação do departamento. Em especial, a arquitetura desenvolvida servirá de plataforma para a pesquisa e caracterização de novos algoritmos de escalonamento.

APÊNDICES

A. ESTRUTURAS DE SUPORTE À INTEGRAÇÃO E DEPURAÇÃO DO SISTEMA

Para a integração e depuração do sistema desenvolvido, foram adotadas algumas estratégias de forma a possibilitar que os testes do *hardware* e do *software* implementados fossem realizados por partes, facilitando a identificação e correção dos problemas que viessem a surgir. Foram estas:

- Previsões de *hardware* para apoio a depuração
- Teste isolado do *hardware* do Bloco Executor
- Teste isolado do *hardware* do Bloco Administrador
- Teste isolado do *hardware* do Bloco de Comunicação
- Teste dos três blocos integrados
- Simulação do *software* do Bloco Executor
- Simulação do *software* do Bloco Administrador
- Simulação do *software* do Bloco de Comunicação
- Teste do *software* e do *hardware* integrados do Bloco Executor e do Bloco Administrador.
- Teste do *software* e do *hardware* integrados de toda arquitetura.

A.1 PREVISÕES DE *HARDWARE* PARA APOIO À DEPURAÇÃO

Por no momento da confecção da placa, não existir no Laboratório de Automação um emulador, equipamento adquirido somente no final de 1997, para facilitar a depuração do *hardware* e do *software* desenvolvido, diversos cuidados foram tomados no projeto da mesma para que os problemas que por ventura pudessem vir a aparecer fossem mais facilmente identificados e corrigidos.

A.1.1 *Jumpers*

Por se tratar da construção de um protótipo que possibilitasse a validação da arquitetura proposta, optou-se pela colocação de diversos *Jumpers* que facilitassem a depuração. Com estes, eventuais problemas de *hardware* ou *software* que fossem enfrentados, poderiam ser rastreados mais facilmente.

Objetivo	Jumper	Conexão
µGerente **recebe** dados de equipamento externo a UP. Usado para teste isolado do Bloco Administrador, realizando monitoração através de um PC conectado pela interface serial RS232.	JP1 JP10	1-2 1-2
µGerente **envia** dados para equipamento externo a UP. Usado para teste isolado do Bloco Administrador, realizando monitoração através de um PC conectado pela interface serial RS232.	JP2 JP9	1-2 1-2
µPrincipal **recebe** dados de equipamento externo a UP. Usado para teste isolado do Bloco Executor, realizando monitoração através de um PC conectado pela interface serial RS232.	JP1 JP10	3-4 1-2
µPrincipal **envia** dados para equipamento externo a UP. Usado para teste isolado do Bloco Executor, realizando monitoração através de um PC conectado pela interface serial RS232.	JP2 JP9	3-4 1-2
Utilização de um bit de controle para a comunicação serial entre o µGerente e o µExecutor. Ver 5.2.	JP3	1-2
Permite utilizar o pino INT1 do µExecutor como entrada de interrupção acessável externamente através do conector CN2.	JP3	Aberto
Estado de Reset	JP4	1-2
Handshake por *Hardware*, interligando RTS da UP com CTS de dispositivo externo com o qual a UP esta se comunicando.	JP5	1-2
Uso do RTS como segundo bit de transmissão da UP. Possibilita *handshake* por *software*.	JP5	2-3
Interface EIA485 com alimentação independente do resto do circuito da UP.	JP6 JP7	1-2 1-2
Utilizar para alimentação do circuito da interface EIA485 a mesma fonte de alimentação do restante da UP. Esta opção é útil quando estamos na fase de depuração e testes da UP, onde queremos determinar a correção quanto à funcionalidade do *hardware/software* implementado.	JP6 JP7	2-3 2-3
Alimentação por bateria nos principais circuitos integrados (chips) da UP. Com esta conexão realizada, diversos chips terão sua alimentação sempre presente. Em caso de falta de energia elétrica na rede pública ou da UP ser desligada, estes chips terão os seus dados preservados, pois terão uma alimentação mínima garantida por uma bateria. Com a cessação do motivo da falta, o processamento pode ser continuado de onde havia parado no instante da interrupção.	JP8	1-2
Não utilizar alimentação por bateria em caso de falta Com esta conexão realizada, todo o circuito ficará sem alimentação caso falte energia	JP8	2-3

elétrica na rede pública ou a UP for desligada. Como conseqüência, quando o motivo da falta cessar, a UP terá que ser reinicializada.		
Operação Normal. Isto é, a UP deve operar com os três microcontroladores ativos e com suas atribuições normais, conforme definido na arquitetura que trata esta dissertação.	JP1 JP2 JP4 JP9 JP10	aberto aberto aberto 2-3 2-3

A.1.2 Conectores

Visando facilitar a visualização dos sinais existentes na placa, para facilitar a identificação e correção de possíveis problemas que viessem a estar ocorrendo, diversos conectores de acesso aos mais importantes sinais existentes foram disponibilizados na periferia da placa confeccionada.

Os dois primeiros conectores seguem um padrão básico de distribuição. Este padrão é composto por dez pinos, sendo oito bits que podem ser os oito bits de uma porta de um dos microcontroladores e mais um pino com o sinal de Vcc e um pino com o sinal de GND. Separando cada módulo que segue este padrão, seguem sempre duas fileiras de dois pinos que não possuem conexão alguma. Estes visam a possibilidade de se colocar cabos separados para cada um dos módulos de dez pinos. Com esta padronização fica facilitada a conexão de cabos nos mesmos e a utilização de chaves e jogos de leds para visualizar algumas informações sobre o status do programa que está sendo executado em um dos microcontroladores.

A.1.2.1 Conector CN1

Conector de trinta e quatro (17 x 2) pinos que contém três vias de Vcc e três vias de GND distribuídos em seus pinos de modo a implementar dois módulos de dez pinos, como descrito anteriormente. Os demais pinos estão sem conexão alguma. A existência deste conector deve-se a uma precaução quanto à possível necessidade que poderia surgir durante a depuração e/ou implementação de uma aplicação na UP, de se conectar alguns sinais externos a algum pino dentro da UP que não está disponível nos outros conectores.

A.1.2.2 Conector CN2

Conector de cinqüenta pinos onde estão dispostos todos os pinos dos três microcontroladores presentes na UP que não estão conectados a nenhum outro componente interno a esta, quais sejam:

- Do µPrincipal está disponível a porta 1 completa, além dos dois pinos de interrupção.

- Do µGerente estão disponíveis os oito sinais destinados aos eventos assíncronos, bem como as duas entradas para os temporizadores/contadores do mesmo.

- Do µComunicação está disponível a porta 2 completa, além de mais três pinos da porta 1. Uma opção de utilização destes pinos seria a conexão da UP a um barramento com transmissão de dados por uma interface paralela.

A.1.2.3 Conector CN3

Para facilitar a depuração, foi colocado um conector de trinta e quatro pinos que contém diversos sinais de controle utilizados na ativação dos componentes presentes na UP. Com isto, ficou facilitado o trabalho de rastreamento dos sinais durante o período de depuração da Unidade de Processamento.

A.1.2.4 Conector CN4

Conector DB9 macho onde está implementada a interface serial RS232.

A.1.2.5 Conector CN5

Conector DB9 fêmea onde está implementada a interface serial EIA485. Foi escolhido a utilização de um conector fêmea, oposto ao CN4, para evitar equívoco na hora de conectar um cabo na interface serial EIA485 ou RS232.

A.1.3 Relógios

Cada um dos microcontroladores utilizados possuem dois pinos chamados XTAL1 e XTAL2 que são, respectivamente, a entrada e a saída de um amplificador inversor , o qual pode ser configurado para prover o circuito do componente com o sinal oscilatório que o mesmo precisa quando for conectado a estes. Outra opção é a desconexão de XTAL2 e o fornecimento em XTAL1 de um sinal oscilatório externo.

Para a UP, foi feita a seguinte montagem: O circuito da Figura A-1 foi colocado nas entradas XTAL1 e XTAL2 do µComunicação, chamadas de MCX1 E MCX2, respectivamente. Deste último pino foi retirado o sinal já oscilante e posto em um *buffer* o qual providenciará um sinal oscilatório externo ao µPrincipal e ao µExecutor.

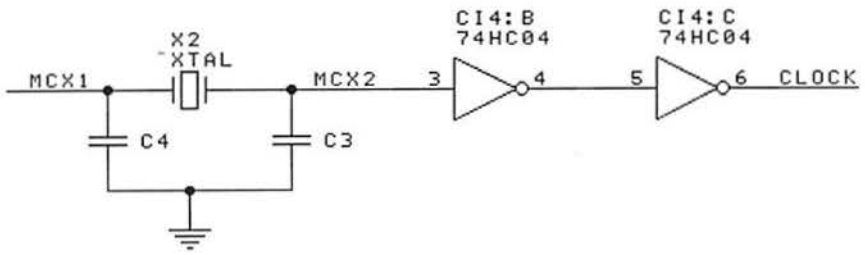


Figura A-A-1 Circuito para fornecimento dos sinais de relógios

A.1.4 Conjunto de Chaves e Leds

Por não estar disponível um emulador para depuração, tornaram-se necessárias mais algumas alternativas que possibilitassem a monitoração e/ou atuação nos programas que estão sendo executados nos microcontroladores. Dentre estas alternativas, um conjunto de chaves Dip-switches e um conjunto de Leds foram de extrema importância para facilitar a depuração dos *softwares* desenvolvidos.

Aproveitando que a grande maioria dos sinais dos microcontroladores foram disponibilizados nos conectores montados, com a utilização de cabos apropriados pode-se, no *hardware* implementado, simular o acontecimento de qualquer um deste sinais através do jogo de chaves Dip-switches colocadas.

O jogo de Leds permite monitorar em que ponto o programa parou ou se perdeu. Para tanto, basta conectar alguma das portas disponíveis no CN2, por exemplo, ao conjunto de Leds e incluir no programa o envio para a respectiva porta um valor de 8 bits diferente em diferentes partes do programa. Dependendo do valor que estiver presente nos Leds, sabe-se até que ponto o programa conseguiu executar sem problemas.

A.1.5 Área de Protótipo

Foi ainda incluído na placa confeccionada para implementar o *hardware* desenvolvido, uma área destinada à inclusão de novos componentes integrados ou discretos. Com isto, se fosse identificada a necessidade de modificações no circuito com a inclusão de novos componentes e/ou a aplicação que se deseja implementar requisite um circuito

eletrônico de apoio, estes poderiam ser feitos nesta área já existente, evitando que uma placa extra tivesse que ser confeccionada.

A.2 TESTE ISOLADO DO *HARDWARE* DO BLOCO EXECUTOR

Para testes do *hardware* implementado, com o Bloco Executor operando isoladamente, foram tomadas as seguintes providências:

- Configuração dos *Jumpers* JP1, JP2, JP9 e JP10 de modo a interligar a serial do µPrincipal diretamente com a serial de um computador PC, possibilitando que se monitorasse informações sobre o andamento do programa.
- Conexão da porta P1, destinada à aplicação do usuário, ao conjunto de chaves Dip-switches de forma a simular a ocorrência de algum sinal externo.
- Escrita de um *software* de teste que realizasse as seguintes tarefas:
    - Escrita de dados na memória Ram externa nos endereços:
        - 2000h à 2010h, área reservada para dados
        - 2000h à 2010h, área reservada para código
        - de 2011h até o fim da área reservada para dados, em intervalos de 1024 bytes.
        - de 2011h até o fim da área reservada para código, em intervalos de 1024 bytes.
    - Leitura dos dados escritos e envio dos mesmos pela serial para que possam ser analisados no computador PC.
    - Verificação periódica da chegada de um evento na Porta 1. Caso percebido um evento, envia o número do pino onde este ocorreu pela serial, precedido pelos caracteres I, N, T.

A.3 TESTE ISOLADO DO *HARDWARE* DO BLOCO ADMINISTRADOR

Para testes do *hardware* implementado, com o Bloco Administrador operando isoladamente, foram tomadas as seguintes providências:

- Configuração dos *Jumpers* JP1, JP2, JP9 e JP10 de modo a interligar a serial do µGerente diretamente com a serial de um computador PC, possibilitando que se monitorasse informações sobre o andamento do programa.

- Conexão da porta P1, destinada à entrada de oito eventos assíncronos, ao conjunto de chaves Dip-switches de forma a simular a ocorrência de algum sinal externo.

- Escrita de um *software* de teste que realizasse as seguintes tarefas:

    - Programa uma data e uma hora no RTC e solicita um aviso para 30 segundos após. Ao receber o aviso, envia um byte pela porta serial, o qual será lido pelo computador PC a este conectado. Este então pergunta o horário para o µG, que envia a informação, após ter solicitado ao RTC.

Periodicamente verifica a chegada de um evento na Porta 1. Caso percebido um evento, envia pela serial o número do pino onde este ocorreu, precedido pelos caracteres ASCII I, N, T.

A.4 TESTE ISOLADO DO *HARDWARE* DO BLOCO DE COMUNICAÇÃO

Para testes do *hardware* implementado, com o Bloco de Comunicação operando isoladamente, foram tomadas as seguintes providências:

- Configuração dos *Jumpers* JP1, JP2, JP9 e JP10 de modo a interligar a serial do µComunicação diretamente com a serial de um computador PC, possibilitando que se monitorasse informações sobre o andamento do programa.

- Conexão da porta P2, destinada a interligação da UP com um barramento paralelo, ao conjunto de chaves Dip-switches de forma a simular a ocorrência de algum sinal externo.

- Escrita de um *software* de teste que realizasse a seguinte tarefa:

    - Periodicamente verifica a chegada de um evento na Porta 2. Caso percebido um evento, envia o número do pino onde este ocorreu pela serial, precedido pelos caracteres I,N,T.

A.5 TESTE DOS TRÊS BLOCOS INTEGRADOS

Para testes do *hardware* implementado com os três microcontroladores trocando informações entre si, as configurações dos *Jumpers* obedeceram o critério de possibilitar a exploração de toda a potencialidade da arquitetura implementada. Além disto, foram implementados os *softwares* necessários para que fosse testada a comunicação entre os três microcontroladores, e entre a UP e o mundo externo utilizando o seguinte algoritmo:

- O µComunicação escreve 32 bytes na FIFO Ram de recebimento de mensagens, com valores seqüenciais de 0 a 32.

- O µComunicação avisa o µGerente da existência destas informações na FIFO Ram de recebimento de mensagens.

- O µGerente por sua vez, repassa a informação ao µPrincipal.

- O µPrincipal então retira os dados da FIFO Ram de recebimento de mensagens, incrementa em uma unidade cada um deles, e coloca-os agora na FIFO Ram de transmissão de mensagens.

- Quando os dados chegam na FIFO Ram de transmissão de mensagens, o *hardware* gera uma interrupção para o µComunicação, indicando a existência de dados nesta memória.

- O µComunicação retira os dados da FIFO e retransmite-os pela serial para o PC ao qual a UP está conectada.

- Os dados então podem ser analisados no PC para ver se estão de acordo com o padrão esperado.

A.6 SIMULAÇÃO DO *SOFTWARE* DO BLOCO EXECUTOR

Pela existência junto ao compilador da Keil *Software* Inc. de um simulador do código gerado, foi possível a detecção com esta ferramenta de diversos problemas de concepção no *software* desenvolvido. Com isto, reduzimos a complexidade da integração do *software* com o *hardware*. Durante a simulação do Bloco Executor, foi testado o seu comportamento e reação quando recebendo ordens simuladas do µGerente para processar diversas tarefas síncronas e assíncronas, além do envio e recebimento de mensagens.

A.7 SIMULAÇÃO DO *SOFTWARE* DO BLOCO ADMINISTRADOR

Durante a simulação do Bloco Administrador, foi testado o seu comportamento e reação quando programado para gerenciar diversas tarefas síncronas e assíncronas, programar o RTC e todas as outras tarefas para este bloco reservadas.

A.8 TESTE DO *SOFTWARE* E DO *HARDWARE* INTEGRADOS DO BLOCO EXECUTOR E DO BLOCO ADMINISTRADOR.

Com a maior confiabilidade no *hardware* e nos *softwares* desenvolvidos, gerada pelos testes anteriores, a junção de ambos para um funcionamento conjunto tornou-se mais tranqüila, pois o números de problemas que ainda precisariam ser resolvidos diminuiu bastante, bem como a localização dos mesmos que ficou restrita a poucas possibilidades, visto que a maioria das partes do *software* e do *hardware* já haviam sido depuradas.

Os testes foram realizados com a implementação do conjunto de tarefas que são utilizadas por uma máquina viradora de couro que utilizava o núcleo de tempo-real desenvolvido pelas Máquinas Kehl S/A Indústria e Comércio.

A.9 TESTE DO *SOFTWARE* E DO *HARDWARE* INTEGRADOS DE TODA ARQUITETURA.

Após a conclusão dos sete primeiros testes, foi incorporado o *software* do µComunicação para a finalização da etapa de testes. Embora este *software* não tivesse sido simulado, em virtude da dificuldade de implementação de uma simulação que fosse útil para depuração, sua incorporação não foi problemática em virtude da simplicidade da estrutura do *software* deste componente.

B. GLOSSÁRIO DE TERMOS

A fim de facilitar a leitura desta dissertação por pessoas que não estejam muito acostumadas aos termos técnicos que envolvem Sistemas Tempo-Real Distribuídos, este apêndice contém um breve resumo de algum destes termos usados no transcorrer deste trabalho.

- **Algoritmo de Escalonamento:** Algoritmo que determina qual entre as tarefas que estão prontas para serem executadas terá acesso primeiro ao processador. (Pg. 8)
- **Algoritmo de Escalonamento Não-Preemptivo:** Algoritmo de escalonamento que não permite a preempção de uma tarefa que esteja executando, mesmo que uma outra tarefa mais prioritária esteja pronta para executar. Justifica-se a não preempção, a fim de minimizar o número de chaveamentos de contexto. (Pg. 8)
- **Algoritmo de Escalonamento Preemptivo:** Algoritmo de escalonamento cuja filosofia dará acesso ao processador a uma tarefa mais prioritária, assim que ela tiver seu requisito de ativação satisfeito, interrompendo uma tarefa menos prioritária que esteja de posse do processador. (Pg. 8)
- **Chaveamento de Contexto:** Ato de salvar o contexto da tarefa que estava executando e restaurar o contexto que a tarefa que vai executar possuía quando foi interrompido pela última vez. Por contexto de uma tarefa entende-se o conteúdo de um conjunto de variáveis, normalmente os registradores do processador, utilizadas pela tarefa para processamento de seu algoritmo. (Pg. 10)
- **Coordenação:** Ação do Sistema Operacional para possibilitar que diferentes tarefas compartilhem objetivos, dados e recursos. (Pg. 10)

- ***Deadline*:** Limite temporal máximo para a execução completa de uma tarefa. (Pg. 10)
- **Determinismo Temporal:** Um sistema é considerado determinístico do ponto de vista temporal, caso os tempos de execução dos processamentos a ele associados sejam conhecidos e finitos. (Pg. 8)
- **FIFO (First Input – First Output):** Estrutura de dados capaz de armazenar um conjunto de variáveis, sendo que o primeiro dado nela inserido, será sempre o primeiro que será retirado. (Pg. 9)
- **Flag de Evento:** Sinalizador binário que indica a ocorrência ou não de um evento. (Pg. 12)
- ***Kernel* (Núcleo):** Considera-se como núcleo de um sistema operacional multi-tarefa a parte responsável pelo gerenciamento das tarefas e comunicação entre estas. (Pg. 10)
- **Polling:** Amostragem periódica. (Pg. 14)
- **Preempção:** Refere-se à interrupção do processamento de uma tarefa menos prioritária que esteja executando, a fim de que o processador seja liberado para outra tarefa mais prioritária que tenha se tornada pronta para executar. (Pg. 8)
- **SIFO (Smallest Input – First Output):** Fila onde os dados nela inseridos são ordenados em ordem crescente dos seus requisitos de ativação temporal. (Pg. 20)
- **Sistemas Distribuídos:** Consistem em uma rede de Unidades de Processamento autônomas, interligadas entre si, onde um protocolo de comunicação é utilizado para interconectá-las. (Pg. 13)
- **Sistemas Tempo-Real:** É aquele cuja corretude depende não apenas do correto processamento lógico de entradas em saídas, mas também da satisfação de requisitos temporais, tais como o instante de tempo no qual as entradas são amostradas e as saídas são geradas. (Pg. 3)
- **Sistemas Tempo-Real Crítico:** É aquele nos quais a não satisfação dos requisitos temporais podem ocasionar desastres e perdas de vidas humanas. (Pg. 4)

- **Sistemas Tempo-Real Não-Crítico:** Semelhante ao sistema tempo-real crítico, mas que tolera eventuais atrasos. (Pg. 4)
- **Tarefas:** É uma atividade que possui um programa, entrada de dados, saída de dados e um estado, podendo ser executada concorrentemente com outras tarefas. (Pg. 7)
- **Tarefas Assíncronas:** Tarefas cujo requisito de ativação é um sinal externo ou uma mensagem cujo instante de ocorrência não pode ser definido a priori. (Pg. 14)
- **Tarefas Síncronas Absolutas:** Tarefas cujos requisitos de ativação são dependentes do tempo e ocorrerão em horários específicos. (Pg. 14)
- **Tarefas Síncronas Cíclicas:** Tarefas cujos requisitos de ativação são dependentes do tempo e periódicos. (Pg. 14)
- **Tarefas Síncronas Relativas:** Tarefas cujos requisitos de ativação são dependentes do tempo, mas o tempo especificado somente começa a ser contado após a ocorrência de um determinado evento. (Pg. 14)

BIBLIOGRAFIA

[BuWe90] Burns A. and Wellings A.J., *Real-Time Systems and Their Programming Languages*, Padstow, Grã-Bretanha, Addison-Wesley Publishers, 1990

[BuWe91] Burns A. and Wellings A.J., *Real-Time Systems*, Padstow, Grã-Bretanha, Addison-Wesley Publishers, 1991

[Gallmeister95] Gallmeister B. O., *Programming for the Real World - Posix.4*, Sebastopol, Estados Unidos, O'Reilly & Associates, 1995

[Ganssle92] Ganssle J.G., *The Art of Programming Embedded Systems*, San Diego, Estados Unidos, Academic Press, 1992

[HaSt91]. Halang W. and Stoyenko A. D., *Constructing Predictable Real Time Systems,* Boston-Dordrecht-London. Kluwer Academic Publishers, 1991.

[HWP95] Halang W., Wannemacher M., and Pereira C.E., *A High-Precision Timing and Interrupt Controller to Support Real-Time Distributed Systems,* 28th Midwest Symposium on Circuits and Systems, Rio de Janeiro, 1995.

[KoOc87] Kopetz H. and Ochsenreiter W., *Clock Synchronization in Distributed Real-Time Systems'*, IEEE Transactions on Computers C-36, 8, pp. 933-940, 1987.

[Labrosse92] Labrosse J.J., *µC/OS - The Real-Time Kernel*, Lawrence, Estados Unidos, R & D Publications, 1992

[LiLa73] Liu C.L., Layland J. W., *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment,*Journal of ACM, Vol. 20, No. 1, pp. 46-61, 1973.

[Motus93] Motus L., *Time Concepts in Real-Time Software*,Control Engineering Practice, Vol. 1, No. 1, pp. 21-33, Oxford, 1993.

[Oliveira97] Oliveira R.S., *Escalonamento de Tarefas Imprecisas em Ambiente Distribuído,* Tese de Doutorado da Universidade Federal de Santa Catarina, Florianópolis, 1997.

[Parisoto97] Parisoto A. at Al., *F-Timer: Dedicated FPGA to Real-Time Systems Design Support*, 9th Euromicro Workshop on Real-Time Systems, Toledo, Espanha, 1997.

[PeFr96] Pereira C.E. e Frigeri A.H., *OORTAC: Um Método de Desenvolvimento de Sistemas de Automação em Tempo-Real Usando Técnicas de Orientação a Objetos,* Anais XI Congresso Brasileiro de Automática, Volume 3 - pp. 1285 - 1290, São Paulo, 1996.

[PFDH96] Pereira C.E., Frigeri A.H., Darscht P. and Halang W., *Object-oriented development of real-time industrial automation systems,* Proc., IFAC Triennial World Congress, San Francisco USA, pp. 321-326. Vol O, Power Plants and Systems, Computer Control, 1996.

[PLH96] Pereira C.E., Lubaszewski M. e Halang W., *Sistema de Suporte ao Escalonamento de Tarefas em Aplicações em Tempo Real Distribuídas com Circuitos de Deteção Concorrente de Erros*, IX Simpósio Brasileiro de Concepção de Circuitos Integrados - SBCCI96, pp. 141-152, Recife, 1996.

[PoPe96] Pontremoli M.M.B. and Pereira C.E., *Proposta de uma Arquitetura de Baixo Custo para Construção de Sistemas Distribuídos Tempo-Real,*Seminário Interno do Departamento de Engenharia Elétrica e da Instrumentação Eletro-Eletrônica, Porto Alegre, pp.92-94, 1996.

[PoPe97a] Pontremoli M.M.B. and Pereira C.E., *Hardware de Alto Desempenho para Automação de Sistemas de Tempo-Real Distribuídos,* Egatea - Revista da Escola de Engenharia da Universidade Federal do Rio Grande do Sul, v. 25, nº.1, Porto Alegre, pp. 73-77, 1997.

[PoPe97b] Pontremoli M.M.B. and Pereira C.E., *Hardware Support For Distributed Real-Time Operating Systems*, Preprints of the 4th IFAC Workshop on Algorithms and Architectures for Real-Time Control, Vila Moura, Portugal, pp. 188-191, 1997.

[PoPe97c] Pontremoli M.M.B. and Pereira C.E., *Hardware Support For Distributed Real-Time Operating Systems*, Special Section of IFAC Control Engineering Practice Journal Vol 5, Number 10, Elsevier Science, pp. 1435-1441, 1997

[PPS98] Pereira C.E., Pontremoli M.M.B. e Santos S.R., *Arquitetura para Automação de Baixo Custo de Sistemas Tempo-Real Distribuídos,* Anais do XII Congresso Brasileiro de Automática, Uberlândia, Brasil, 1998.

[Ripps93] Ripps D.L., *Guia de Implementação para programação em Tempo Real*, tradução de Caetano Loiola, Rio de Janeiro, Brasil, Campus, 1993

[SiSw82] Siewiorek P. and Swarz R. S., *The Theory and Practice of Reliable System Design*, Digital Press, 1982.

[Souza97] Souza Jr. A. et Al., *Design Alternatives in F-Timer: ASIC to Real-Time Systems Support*, Tercer Taller Iberoamericano de Microelectrónica y sus Aplicaciones, México D.F., México, pp. 209-217, 1997.

[Stankovic88] Stankovic J., *Misconceptions about real-time computing: a serius problem for next-generation systems.* Computers, 1988.

[StRa92] Stankovic J. and Ramaamritham K., *Advaances in Real-Time Systems*, Los Alamitos, Estados Unidos, IEEE Computer Society Press, 1992.

[Tanenbaum92] Tanenbaum A.S., *Modern Operating Systems,* Prentice Hall International, New Jersey, 1992.

[VoMu86] Volz R.A. and Mudge T.N., *Instruction Level Mechanisms for Accurate Real-Time Task Scheduling*, Proc. IEEE Real-Time Systems Symposium, pp. 209-215, New Orleans, 1986.

[Young82] Young S.J., *Real Time Languages: Design and Development.* Chichester: Ellis Horwood, 1982